aws

Developer Guide

Amazon CloudFront



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon CloudFront: Developer Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the AWS European Sovereign Cloud User Guide.

Table of Contents

What is Amazon CloudFront? 1
How you set up CloudFront to deliver content 2
Choose between standard distribution or multi-tenant distribution 4
Pricing 4
Ways to use CloudFront 5
Accelerate static website content delivery 5
Serve video on demand or live streaming video 5
Encrypt specific fields throughout system processing 6
Customize at the edge 6
Serve private content by using Lambda@Edge customizations 6
How CloudFront delivers content 7
How CloudFront delivers content to your users 7
How CloudFront works with regional edge caches 8
CloudFront edge servers 11
Use the CloudFront managed prefix list 11
Working with AWS SDKs 12
CloudFront technical resources 12
Get started 13
Set up your AWS account 13
Sign up for an AWS account 14
Choose how to access CloudFront 14
Get started with a standard distribution 14
Prerequisites 15
Create bucket 15
Upload content 16
Create distribution 16
Access the content 18
Clean up 18
Enhance your basic distribution 18
Get started (AWS CLI) 19
Prerequisites 19
Create an Amazon S3 bucket 20
Upload the content to the bucket 20
Create an Origin Access Control (OAC) 21

Create a standard distribution ... 21
Update your S3 bucket policy ... 22
Confirm distribution deployment ... 23
Access your content through CloudFront ... 24
Clean up ... 24
Get started with a secure static website ... 26
Solution overview ... 26
Deploy the solution ... 27
E-commerce distributions ... 33
Architecture overview ... 33
Cache behaviors for e-commerce ... 34
TTL strategy for marketplace content ... 36
Origin configuration ... 37
Amazon S3 origin for static assets ... 37
ALB origin for dynamic content ... 37
Application-level caching with ElastiCache (Valkey) ... 38
Frequently asked questions ... 39
How do I handle personalized content with caching? ... 39
How do I choose between invalidation and short TTLs? ... 40
How do I prepare for flash sales and traffic spikes? ... 40
Should I use Origin Shield for my marketplace? ... 40
E-commerce security checklist ... 41
Signed URLs and cookies for premium content ... 41
Origin access control (OAC) for Amazon S3 ... 42
Encryption in transit ... 42
VPC security groups for the cache cluster ... 43
Additional security controls ... 44
CloudFront flat-rate pricing plans ... 46
Benefits of CloudFront flat-rate pricing plans ... 47
Features by pricing plan tier ... 47
Pricing plan features ... 48
Monthly usage allowances ... 84
What counts toward your usage allowance ... 85
What happens when usage exceeds the allowance ... 85
Configurable usage allowances on the Premium plan ... 86
Eligibility based on historical usage ... 87

Costs covered by your plan ... 87
Route 53 DNS management and your plan ... 88
Reduce overall AWS costs with pricing plans ... 89
Manage your flat-rate pricing plans ... 90
Subscribe a new distribution to a pricing plan ... 90
Subscribe an existing distribution to a pricing plan ... 91
Upgrade a pricing plan ... 91
Change your Premium plan usage level ... 92
Downgrade a pricing plan ... 92
Cancel a pricing plan ... 93
Cancel a pending plan change ... 94
Deleting a distribution with a pricing plan ... 94
Permissions ... 95
Flat-rate pricing plan quotas ... 95
Unsupported features ... 95
Unsupported features ... 96
Unsupported associations ... 97
Account-level constraints ... 98
Resource-level constraints ... 98
Additional features that can affect your pricing plan ... 98
Pricing plans versus pay-as-you-go pricing ... 99
Configure distributions ... 101
Understand how multi-tenant distributions work ... 103
How it works ... 104
Terms ... 106
Unsupported features ... 108
Distribution tenant customizations ... 109
Request certificates (distribution tenant) ... 113
Serve distribution tenant domains without a certificate ... 121
Create custom connection group (optional) ... 125
Address-based selection ... 126
Viewer fallback ... 135
Migrate to a multi-tenant distribution ... 138
Create a distribution ... 140
Create a CloudFront distribution in the console ... 141
Values that are displayed ... 147

Additional links ... 148
Add a domain to your CloudFront standard distribution ... 148
Preconfigured distribution settings ... 150
Amazon S3 origin ... 151
API Gateway origin ... 152
Custom origin and EC2 instance ... 153
Elastic Load Balancing origin ... 155
MediaPackage v1 origin ... 157
MediaPackage v2 origin ... 158
MediaTailor origin ... 159
All distribution settings ... 160
Origin settings ... 161
Cache behavior settings ... 173
Distribution settings ... 187
Custom error pages and error caching ... 196
Geographic restrictions ... 197
Test a distribution ... 198
Create links to your objects ... 198
Update a distribution ... 199
Update a distribution in the console ... 199
Tag a CloudFront resource ... 202
Tag restrictions ... 203
Tag your resources for billing ... 203
Add, edit, and delete tags ... 204
Programmatic tagging ... 205
Delete a distribution ... 205
Use various origins ... 207
Use an Amazon S3 bucket ... 208
Use a MediaStore container or a MediaPackage channel ... 221
Use an Application Load Balancer ... 221
Use a Network Load Balancer ... 221
Use a Lambda function URL ... 222
Use an AWS Lambda Web Function ... 223
Use Amazon EC2 (or another custom origin) ... 224
Use CloudFront origin groups ... 226
Use Amazon API Gateway ... 226

Enable IPv6 ... 226
IPv6 viewer requests ... 227
IPv6 origin requests ... 228
Use continuous deployment to safely test changes ... 228
CloudFront continuous deployment workflow ... 231
Work with a staging distribution and continuous deployment policy ... 232
Monitor a staging distribution ... 242
Learn how continuous deployment works ... 243
Quotas and other considerations for continuous deployment ... 245
Use custom URLs ... 246
Requirements for using alternate domain names ... 246
Restrictions on using alternate domain names ... 248
Add an alternate domain name ... 251
Move an alternate domain name ... 254
Remove an alternate domain name ... 275
Use wildcards in alternate domain names ... 277
Use WebSockets ... 277
How the WebSocket protocol works ... 278
WebSocket requirements ... 278
Recommended WebSocket headers ... 279
HTTP versions for WebSocket connections ... 279
Request Anycast static IPs to use for allowlisting ... 280
Prerequisites ... 280
Request an Anycast static IP list ... 280
Create an Anycast static IP list ... 281
Associate an Anycast static IP list with an existing distribution ... 281
Associate an Anycast static IP list with a new distribution ... 282
Associate an Anycast static IP list with a connection group ... 283
Update an Anycast static IP list ... 283
Bring your own IP to CloudFront using IPAM ... 284
Using gRPC ... 289
How gRPC works in CloudFront ... 289
Working with shared resources in CloudFront ... 291
Prerequisites for sharing resources ... 291
Sharing a VPC origin ... 292
Using a shared VPC origin ... 295

Identifying a shared VPC origin ... 295
Unsharing a shared VPC origin ... 296
Responsibilities and permissions for shared VPC origins ... 297
Permissions for owners ... 297
Permissions for consumers ... 297
AWSRAMDefaultPermissionCloudFront ... 297
Billing and metering ... 297
Shared resource quotas ... 297
Caching and availability ... 299
Improve your cache hit ratio ... 300
Specify how long CloudFront caches your objects ... 300
Use Origin Shield ... 300
Caching based on query string parameters ... 301
Caching based on cookie values ... 301
Caching based on request headers ... 302
Remove `Accept-Encoding` header when compression is not needed ... 303
Serve media content over HTTP ... 304
Use Origin Shield ... 304
Use cases for Origin Shield ... 305
Choose the AWS Region for Origin Shield ... 310
Enable Origin Shield ... 312
Estimate Origin Shield costs ... 314
Origin Shield high availability ... 315
How Origin Shield interacts with other CloudFront features ... 315
Increase availability with origin failover ... 316
Create an origin group ... 318
Control origin timeouts and attempts ... 319
Use origin failover with Lambda@Edge functions ... 321
Use custom error pages with origin failover ... 322
Manage cache expiration ... 322
Use headers to control cache duration for individual objects ... 323
Serve stale (expired) content ... 325
Specify the amount of time that CloudFront caches objects ... 327
Add headers to your objects by using the Amazon S3 console ... 333
Caching and query string parameters ... 333
Console and API settings for query string forwarding and caching ... 335

Optimize caching ... 336
Query string parameters and CloudFront standard logs (access logs) ... 337
Cache content based on cookies ... 338
Cache content based on request headers ... 341
Headers and distributions – overview ... 342
Select the headers to base caching on ... 343
Configure CloudFront to respect CORS settings ... 344
Configure caching based on the device type ... 345
Configure caching based on the language of the viewer ... 345
Configure caching based on the location of the viewer ... 345
Configure caching based on the protocol of the request ... 345
Configure caching for compressed files ... 346
How caching based on headers affects performance ... 346
How the case of headers and header values affects caching ... 346
Headers that CloudFront returns to the viewer ... 346
Control the cache key with a policy ... 347
Understand cache policies ... 348
Policy information ... 348
Time to live (TTL) settings ... 348
Cache key settings ... 349
Create cache policies ... 355
Use managed cache policies ... 359
Amplify ... 360
CachingDisabled ... 361
CachingOptimized ... 362
CachingOptimizedForUncompressedObjects ... 363
Elemental-MediaPackage ... 363
UseOriginCacheControlHeaders ... 364
UseOriginCacheControlHeaders-QueryStrings ... 365
Understand the cache key ... 366
Default cache key ... 367
Customize the cache key ... 368
Control origin requests with a policy ... 370
Understand origin request policies ... 371
Policy information ... 371
Origin request settings ... 371

Create origin request policies ... 373
Use managed origin request policies ... 378
AllViewer ... 379
AllViewerAndCloudFrontHeaders-2022-06 ... 379
AllViewerExceptHostHeader ... 381
CORS-CustomOrigin ... 382
CORS-S3Origin ... 382
Elemental-MediaTailor-PersonalizedManifests ... 383
HostHeaderOnly ... 383
UserAgentRefererHeaders ... 384
Add CloudFront request headers ... 384
Device type headers ... 385
Viewer location headers ... 386
Headers to determine the viewer's header structure ... 387
TLS-related headers ... 387
Other CloudFront headers ... 390
Understand how origin request policies and cache policies work together ... 391
Add or remove response headers with a policy ... 394
Understand response headers policies ... 395
Policy details (metadata) ... 395
CORS headers ... 396
Security headers ... 399
Custom headers ... 402
Remove headers ... 402
Server-Timing header ... 404
Create response headers policies ... 409
Use managed response headers policies ... 415
CORS-and-SecurityHeadersPolicy ... 416
CORS-With-Preflight ... 417
CORS-with-preflight-and-SecurityHeadersPolicy ... 417
SecurityHeadersPolicy ... 418
SimpleCORS ... 419
Request and response behavior ... 421
How CloudFront processes HTTP and HTTPS requests ... 421
Request and response behavior for Amazon S3 origins ... 422
How CloudFront processes and forwards requests to your Amazon S3 origin ... 422

How CloudFront processes responses from your Amazon S3 origin ... 429
Request and response behavior for custom origins ... 431
How CloudFront processes and forwards requests to your custom origin ... 431
How CloudFront processes responses from your custom origin ... 449
Request and response behavior for origin groups ... 453
Add custom headers to origin requests ... 454
Use cases ... 454
Configure CloudFront to add custom headers to origin requests ... 455
Custom headers that CloudFront can't add to origin requests ... 456
Configure CloudFront to forward the `Authorization` header ... 457
How CloudFront processes range GETs ... 458
Use range requests to cache large objects ... 459
How CloudFront processes HTTP 3xx status codes from your origin ... 460
How CloudFront processes HTTP 4xx and 5xx status codes from your origin ... 460
How CloudFront processes errors when you have configured custom error pages ... 461
How CloudFront processes errors if you haven't configured custom error pages ... 464
HTTP 4xx and 5xx status codes that CloudFront caches ... 466
Generate custom error responses ... 468
Configure error response behavior ... 468
Create a custom error page for specific HTTP status codes ... 470
Store objects and custom error pages in different locations ... 472
Change response codes returned by CloudFront ... 472
Control how long CloudFront caches errors ... 473
Add, remove, or replace content ... 475
Add and access content ... 475
Use file versioning to update or remove existing content ... 476
Update existing files using versioned file names ... 476
Remove content so CloudFront won't distribute it ... 476
Customize file URLs ... 477
Use your own domain name (example.com) ... 478
Use a trailing slash (/) in URLs ... 478
Create signed URLs for restricted content ... 478
Specify a default root object ... 479
How to specify a default root object ... 479
How default root object works ... 481
How CloudFront works if you don't define a root object ... 482

Invalidate files to remove content ... 483
Choose between invalidating files and using versioned file names ... 483
Determine which paths to invalidate ... 484
What you need to know when invalidating paths ... 484
Invalidating content by cache tags ... 488
What you need to know when invalidating tags ... 493
Invalidate files ... 494
Pay for file invalidation ... 498
Serve compressed files ... 499
Configure CloudFront to compress objects ... 499
How CloudFront compression works ... 500
Conditions for compression ... 501
File types that CloudFront compresses ... 503
ETag header conversion ... 505
Use AWS WAF protections ... 506
Enable AWS WAF for distributions ... 507
Enable AWS WAF for a new distribution ... 507
Use an existing web ACL ... 508
Enable bot control ... 509
Configure protection by bot category ... 509
Manage AWS WAF security protections for CloudFront ... 510
Prerequisites ... 511
Enable AWS WAF logs ... 512
Set up rate limiting ... 512
Disable AWS WAF security protections ... 514
Configure secure access and restrict access to content ... 516
Use HTTPS with CloudFront ... 517
Require HTTPS between viewers and CloudFront ... 518
Require HTTPS to a custom origin ... 520
Require HTTPS to an Amazon S3 origin ... 523
Supported protocols and ciphers between viewers and CloudFront ... 525
Supported protocols and ciphers between CloudFront and the origin ... 534
Use alternate domain names and HTTPS ... 536
Choose how CloudFront serves HTTPS requests ... 537
Requirements for using SSL/TLS certificates with CloudFront ... 541

Quotas on using SSL/TLS certificates with CloudFront (HTTPS between viewers and CloudFront only) ... 545
Configure alternate domain names and HTTPS ... 547
Determine the size of the public key in an SSL/TLS RSA certificate ... 551
Increase the quotas for SSL/TLS certificates ... 552
Rotate SSL/TLS certificates ... 553
Revert from a custom SSL/TLS certificate to the default CloudFront certificate ... 554
Switch from a custom SSL/TLS certificate with dedicated IP addresses to SNI ... 556
Mutual TLS authentication with CloudFront (Viewer mTLS) ... 556
How it works ... 557
Use cases ... 557
Trust stores and certificate management ... 558
Enable mutual TLS for CloudFront distributions ... 565
Associate a CloudFront Connection Function ... 569
Configuring additional settings ... 575
Viewer mTLS headers for cache policies and forwarded to origin ... 577
Certificate revocation ... 581
Helper methods for mutual TLS ... 590
Additional validation modes ... 590
Connection logs for mutual TLS ... 594
Origin mutual TLS with CloudFront ... 594
Viewer mTLS vs Origin mTLS ... 594
How it works ... 595
Use cases ... 595
Important: Origin Server Requirements ... 595
Getting started ... 596
Certificate management with AWS Certificate Manager ... 596
Enable origin mutual TLS for CloudFront distributions ... 600
Using CloudFront Functions with origin mutual TLS ... 603
Restrict content with signed URLs and signed cookies ... 606
How to serve private content ... 606
Restrict access to files ... 607
Specify trusted signers ... 610
Decide to use signed URLs or signed cookies ... 619
Use signed URLs ... 620
Use signed cookies ... 643

Linux commands and OpenSSL for base64 encoding and encryption 674
Code examples for signed URLs 675
Restrict access to an AWS origin 705
Restrict access to an AWS Elemental MediaPackage v2 origin 706
Restrict access to an AWS Elemental MediaStore origin 713
Restrict access to an AWS Lambda function URL origin 721
Restrict access to an AWS Lambda Web Function origin 731
Restrict access to an Amazon S3 origin 737
Restrict access with VPC origins 753
Restrict access to an Amazon S3 Multi-Region Access Point origin 760
Restrict access to Application Load Balancers 767
Configure CloudFront to add a custom HTTP header to requests 768
Configure an Application Load Balancer to only forward requests that contain a specific header 770
(Optional) Improve the security of this solution 771
(Optional) Limit access to origin by using the AWS-managed prefix list for CloudFront 773
Geographic restriction 773
Use CloudFront geographic restrictions 774
Use a third-party geolocation service 776
Use field-level encryption to help protect sensitive data 777
Overview of field-level encryption 779
Set up field-level encryption 780
Decrypt data fields at your origin 786
Video on demand and live streaming video 789
About streaming video 789
Deliver video on demand 790
Configure video on demand for Microsoft Smooth Streaming 791
Deliver video streaming 793
Serve video by using AWS Elemental MediaStore as the origin 794
Serve live video formatted with AWS Elemental MediaPackage 795
Serve video-on-demand content with AWS Elemental MediaPackage 800
Media quality-aware resiliency 806
MQAR log fields 808
Use functions to customize at the edge 809
Differences between CloudFront Functions and Lambda@Edge 810
Customize with CloudFront Functions 812

Tutorial: Create a simple CloudFront function ... 813
Tutorial: Create a CloudFront function that uses key values ... 817
Write function code ... 819
Create functions ... 935
Test functions ... 939
Update functions ... 944
Publish functions ... 946
Associate functions with distributions ... 948
CloudFront KeyValueStore ... 952
Customize with CloudFront Connection Functions ... 973
Overview and workflow ... 974
Configuration and limits ... 976
Create CloudFront Connection Functions for mutual TLS (viewer) validation ... 977
Write CloudFront Connection Function code for mutual TLS (viewer) validation ... 980
Test CloudFront Connection Functions before deployment ... 990
Associate Connection Functions with distributions ... 992
Implement certificate revocation for mutual TLS (viewer) with CloudFront Functions and KeyValueStore ... 993
Customize with Lambda@Edge ... 1000
How Lambda@Edge works with requests and responses ... 1001
Ways to use Lambda@Edge ... 1001
Get started with Lambda@Edge ... 1002
Set up IAM permissions and roles ... 1010
Write and create a Lambda@Edge function ... 1017
Add triggers for a Lambda@Edge function ... 1022
Test and debug ... 1029
Delete functions and replicas ... 1036
Event structure ... 1037
Work with requests and responses ... 1054
Example functions ... 1060
Restrictions on edge functions ... 1099
Restrictions on all edge functions ... 1099
Restrictions on CloudFront Functions ... 1105
Restrictions on Lambda@Edge ... 1108
Reports, metrics, and logs ... 1113
AWS billing and usage reports for CloudFront ... 1113

View the AWS billing report for CloudFront ... 1114
View the AWS usage report for CloudFront ... 1115
Interpret your AWS bill and usage reports for CloudFront ... 1117
View CloudFront console reports ... 1123
View CloudFront cache statistics reports ... 1124
View CloudFront popular objects reports ... 1130
View CloudFront top referrers reports ... 1136
View CloudFront usage reports ... 1139
View CloudFront viewers reports ... 1147
Monitor CloudFront metrics with Amazon CloudWatch ... 1159
View CloudFront and edge function metrics ... 1160
Create alarms ... 1167
Download metrics data ... 1168
CloudFront metrics ... 1171
CloudFront and edge function logging ... 1177
Logging requests ... 1177
Logging edge functions ... 1179
Logging service activity ... 1179
Access logs (standard logs) ... 1179
Use real-time access logs ... 1227
Connection logs ... 1250
Edge function logs ... 1257
AWS CloudTrail logs ... 1261
Track configuration changes with AWS Config ... 1274
Set up AWS Config with CloudFront ... 1274
View CloudFront configuration history ... 1275
Evaluate CloudFront configurations with AWS Config Rules ... 1276
Security ... 1278
Data protection ... 1279
Encryption in transit ... 1280
Encryption at rest ... 1281
Restrict access to content ... 1281
Identity and Access Management ... 1282
Audience ... 1283
Authenticating with identities ... 1283
Managing access using policies ... 1284

How Amazon CloudFront works with IAM ... 1286
Identity-based policy examples ... 1291
AWS managed policies ... 1302
Use service-linked roles ... 1311
Troubleshoot CloudFront identity and access ... 1315
Logging and monitoring ... 1317
Compliance validation ... 1318
CloudFront compliance best practices ... 1319
Resilience ... 1320
CloudFront origin failover ... 1321
Infrastructure security ... 1321
Troubleshooting ... 1323
Troubleshooting distribution issues ... 1323
CloudFront returns an Access Denied error ... 1323
CloudFront returns an InvalidViewerCertificate error when I try to add an alternate domain name ... 1326
CloudFront returns an incorrectly configured DNS record error when I try to add a new CNAME ... 1327
I can't view the files in my distribution ... 1327
Error message: Certificate: <certificate-id> is being used by CloudFront ... 1329
Troubleshooting error response status codes ... 1330
HTTP 400 status code (Bad Request) ... 1330
HTTP 401 status code (Unauthorized) ... 1332
HTTP 403 status code (Invalid method) ... 1333
HTTP 403 status code (Permission Denied) ... 1333
HTTP 404 status code (Not Found) ... 1335
HTTP 412 status code (Precondition Failed) ... 1336
HTTP 500 status code (Internal Server Error) ... 1336
HTTP 502 status code (Bad Gateway) ... 1337
HTTP 503 status code (Service Unavailable) ... 1342
HTTP 504 status code (Gateway Timeout) ... 1345
Load testing CloudFront ... 1350
Quotas ... 1351
General quotas ... 1352
General quotas on distributions ... 1353
General quotas on policies ... 1355

Quotas on mTLS and trust stores ... 1357
Quotas on CloudFront Functions ... 1358
Quotas on CloudFront Functions network calls ... 1358
Quotas on Connection Functions ... 1359
Quotas on key value stores ... 1359
Quotas on Lambda@Edge ... 1360
Quotas on SSL certificates ... 1362
Quotas on invalidations ... 1363
Quotas on key groups ... 1363
Quotas on WebSocket connections ... 1364
Quotas on field-level encryption ... 1364
Quotas on cookies (legacy cache settings) ... 1365
Quotas on query strings (legacy cache settings) ... 1366
Quotas on headers ... 1366
Quotas on multi-tenant distributions ... 1367
Related information ... 1368
Code examples ... 1369
Basics ... 1371
Actions ... 1371
Scenarios ... 1444
Create a multi-tenant distribution and distribution tenant ... 1444
Delete signing resources ... 1455
Get started with a basic content distribution network ... 1457
Getting started with web application firewalls ... 1468
Sign URLs and cookies ... 1481
CloudFront Functions examples ... 1484
Add HTTP security headers ... 1486
Add a CORS header ... 1487
Add a cache control header ... 1488
Add a true client IP header ... 1489
Add an origin header ... 1489
Add index.html to request URLs ... 1490
Authorize before the cache ... 1491
Authorize with a network call ... 1494
Bot check that fails open ... 1498
Health check and failover ... 1502

KeyValueStore read-through config 1507
Make a network call 1510
Markdown for AI agents on demand 1512
Normalize query string parameters 1516
Redirect to a new URL 1517
Rewrite a request URI 1518
Route through a VPC origin 1520
Select origin closer to the viewer 1523
Send events with cf.beacon 1525
Sticky A/B variant 1529
Use key-value pairs 1534
Validate a Cognito token 1535
Validate a simple token 1539
Virtual waiting room 1542
Document history 1550

What is Amazon CloudFront?

Amazon CloudFront is a web service that speeds up distribution of your static and dynamic web content, such as .html, .css, .js, and image files, to your users. CloudFront delivers your content through a worldwide network of data centers called edge locations. When a user requests content that you're serving with CloudFront, the request is routed to the edge location that provides the lowest latency (time delay), so that content is delivered with the best possible performance.

- If the content is already in the edge location with the lowest latency, CloudFront delivers it immediately.
- If the content is not in that edge location, CloudFront retrieves it from an origin that you've defined—such as an Amazon S3 bucket, a MediaPackage channel, or an HTTP server (for example, a web server) that you have identified as the source for the definitive version of your content.

As an example, suppose that you're serving an image from a traditional web server, not from CloudFront. For example, you might serve an image, sunsetphoto.png, using the URL `https://example.com/sunsetphoto.png`.

Your users can easily navigate to this URL and see the image. But they probably don't know that their request is routed from one network to another—through the complex collection of interconnected networks that comprise the internet—until the image is found.

CloudFront speeds up the distribution of your content by routing each user request through the AWS backbone network to the edge location that can best serve your content. Typically, this is a CloudFront edge server that provides the fastest delivery to the viewer. Using the AWS network dramatically reduces the number of networks that your users' requests must pass through, which improves performance. Users get lower latency—the time it takes to load the first byte of the file—and higher data transfer rates.

You also get increased reliability and availability because copies of your files (also known as *objects)* are now held (or cached) in multiple edge locations around the world.

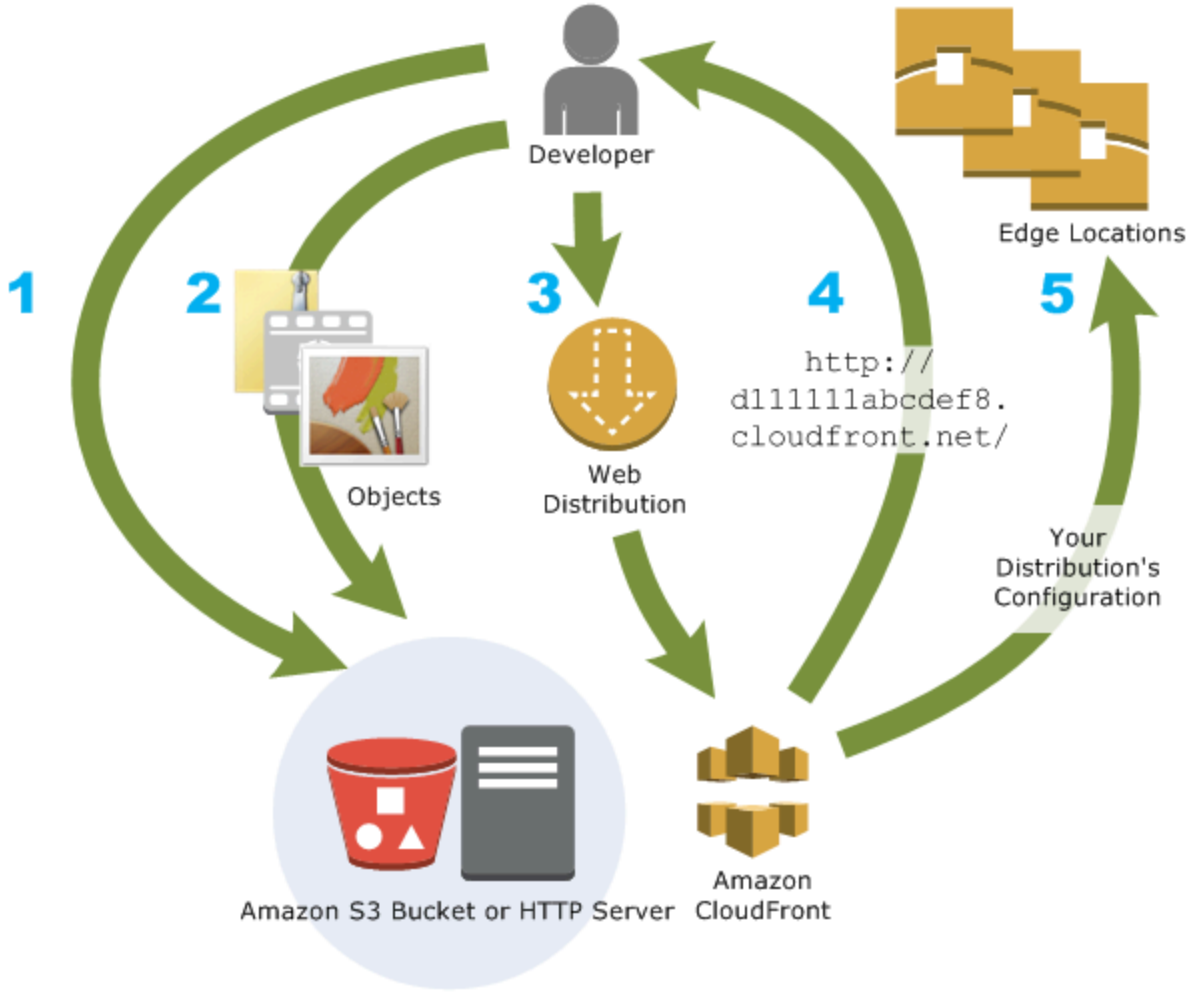
Topics

- How you set up CloudFront to deliver content
- Choose between standard distribution or multi-tenant distribution

- Pricing
- Ways to use CloudFront
- How CloudFront delivers content
- Locations and IP address ranges of CloudFront edge servers
- Using CloudFront with an AWS SDK
- CloudFront technical resources

How you set up CloudFront to deliver content

You create a CloudFront distribution to tell CloudFront where you want content to be delivered from, and the details about how to track and manage content delivery. Then CloudFront uses computers—edge servers—that are close to your viewers to deliver that content quickly when someone wants to see it or use it.



How you configure CloudFront to deliver your content

1. You specify *origin servers*, like an Amazon S3 bucket or your own HTTP server, from which CloudFront gets your files which will then be distributed from CloudFront edge locations all over the world.

   An origin server stores the original, definitive version of your objects. If you're serving content over HTTP, your origin server is either an Amazon S3 bucket or an HTTP server, such as a web server. Your HTTP server can run on an Amazon Elastic Compute Cloud (Amazon EC2) instance or on a server that you manage; these servers are also known as *custom origins.*

2. You upload your files to your origin servers. Your files, also known as *objects*, typically include web pages, images, and media files, but can be anything that can be served over HTTP.

   If you're using an Amazon S3 bucket as an origin server, you can make the objects in your bucket publicly readable, so that anyone who knows the CloudFront URLs for your objects can access them. You also have the option of keeping objects private and controlling who accesses them. See Serve private content with signed URLs and signed cookies.

3. You create a CloudFront *distribution*, which tells CloudFront which origin servers to get your files from when users request the files through your web site or application. At the same time, you specify details such as whether you want CloudFront to log all requests and whether you want the distribution to be enabled as soon as it's created.

4. CloudFront assigns a domain name to your new distribution that you can see in the CloudFront console, or that is returned in the response to a programmatic request, for example, an API request. If you like, you can add an alternate domain name to use instead.

5. CloudFront sends your distribution's configuration (but not your content) to all of its *edge locations* or *points of presence* (POPs)— collections of servers in geographically-dispersed data centers where CloudFront caches copies of your files.

As you develop your website or application, you use the domain name that CloudFront provides for your URLs. For example, if CloudFront returns `d111111abcdef8.cloudfront.net` as the domain name for your distribution, the URL for logo.jpg in your Amazon S3 bucket (or in the root directory on an HTTP server) is `https://d111111abcdef8.cloudfront.net/logo.jpg`.

Or you can set up CloudFront to use your own domain name with your distribution. In that case, the URL might be `https://www.example.com/logo.jpg`.

Optionally, you can configure your origin server to add headers to the files, to indicate how long you want the files to stay in the cache in CloudFront edge locations. By default, each file stays in an edge location for 24 hours before it expires. The minimum expiration time is 0 seconds; there isn't a maximum expiration time. For more information, see Manage how long content stays in the cache (expiration).

Choose between standard distribution or multi-tenant distribution

CloudFront offers distribution options for single websites or apps, and for multi-tenant scenarios.

Standard distribution

Designed for unique configurations per website or application. Choose this in the following use cases:

- You need a standalone CloudFront distribution
- Each site or application requires its own custom settings

Most people start with a standard distribution.

Multi-tenant distribution and distribution tenants (CloudFront SaaS Manager)

Designed specifically for SaaS providers and multi-tenant scenarios. Choose this in the following use cases:

- You're building a SaaS platform to serve multiple customer websites or applications
- You need to manage multiple similar distributions efficiently
- You want centralized control over shared configurations

For more information, see Understand how multi-tenant distributions work.

Pricing

CloudFront charges for data transfers out from its edge locations, along with HTTP or HTTPS requests. Pricing varies by usage type, geographical region, and feature selection.

The data transfer from your origin to CloudFront is always free when using AWS origins like Amazon Simple Storage Service (Amazon S3), Elastic Load Balancing, or Amazon API Gateway.

You are only billed for the outbound data transfer from CloudFront to the viewer when using AWS origins.

For more information, see CloudFront pricing and the Billing and Savings Bundle FAQs.

Ways to use CloudFront

Using CloudFront can help you accomplish a variety of goals. This section lists just a few, together with links to more information, to give you an idea of the possibilities.

Topics

- Accelerate static website content delivery
- Serve video on demand or live streaming video
- Encrypt specific fields throughout system processing
- Customize at the edge
- Serve private content by using Lambda@Edge customizations

Accelerate static website content delivery

CloudFront can speed up the delivery of your static content (for example, images, style sheets, JavaScript, and so on) to viewers across the globe. By using CloudFront, you can take advantage of the AWS backbone network and CloudFront edge servers to give your viewers a fast, safe, and reliable experience when they visit your website.

A simple approach for storing and delivering static content is to use an Amazon S3 bucket. Using S3 together with CloudFront has a number of advantages, including the option to use origin access control to easily restrict access to your Amazon S3 content.

For more information about using Amazon S3 together with CloudFront, including a CloudFormation template to help you get started quickly, see Get started with a secure static website.

Serve video on demand or live streaming video

CloudFront offers several options for streaming your media to global viewers—both pre-recorded files and live events.

- For video on demand (VOD) streaming, you can use CloudFront to stream in common formats such as MPEG DASH, Apple HLS, Microsoft Smooth Streaming, and CMAF, to any device.
- For broadcasting a live stream, you can cache media fragments at the edge, so that multiple requests for the manifest file that delivers the fragments in the right order can be combined, to reduce the load on your origin server.

For more information about how to deliver streaming content with CloudFront, see Video on demand and live streaming video with CloudFront.

Encrypt specific fields throughout system processing

When you configure HTTPS with CloudFront, you already have secure end-to-end connections to origin servers. When you add field-level encryption, you can protect specific data throughout system processing in addition to HTTPS security, so that only certain applications at your origin can see the data.

To set up field-level encryption, you add a public key to CloudFront, and then specify the set of fields that you want to be encrypted with the key. For more information, see Use field-level encryption to help protect sensitive data.

Customize at the edge

Running serverless code at the edge opens up a number of possibilities for customizing the content and experience for viewers, at reduced latency. For example, you can return a custom error message when your origin server is down for maintenance, so viewers don't get a generic HTTP error message. Or you can use a function to help authorize users and control access to your content, before CloudFront forwards a request to your origin.

Using Lambda@Edge with CloudFront enables a variety of ways to customize the content that CloudFront delivers. To learn more about Lambda@Edge and how to create and deploy functions with CloudFront, see Customize at the edge with Lambda@Edge. To see a number of code samples that you can customize for your own solutions, see Lambda@Edge example functions.

Serve private content by using Lambda@Edge customizations

Using Lambda@Edge can help you configure your CloudFront distribution to serve private content from your own custom origin, in addition to using signed URLs or signed cookies.

To serve private content using CloudFront, you do the following:

- Require that your users (viewers) access content using signed URLs or signed cookies.
- Restrict access to your origin so that it's only available from CloudFront origin-facing servers. To do this, you can do one of the following:
  - For an Amazon S3 origin, you can use an origin access control (OAC).
  - For a custom origin, you can do the following:
    - If the custom origin is protected by an Amazon VPC security group or AWS Firewall Manager, you can use the CloudFront managed prefix list to allow inbound traffic to your origin from only CloudFront origin-facing IP addresses.
    - Use a custom HTTP header to restrict access to only requests from CloudFront. For more information, see the section called “Restrict access to files on custom origins” and the section called “Add custom headers to origin requests”. For an example that uses a custom header to restrict access to an Application Load Balancer origin, see the section called “Restrict access to Application Load Balancers”.
    - If the custom origin requires custom access control logic, you can use Lambda@Edge to implement that logic, as described in this blog post: Serving Private Content Using Amazon CloudFront & Lambda@Edge.

How CloudFront delivers content

After some initial setup, CloudFront works together with your website or application and speeds up delivery of your content. This section explains how CloudFront serves your content when viewers request it.

Topics

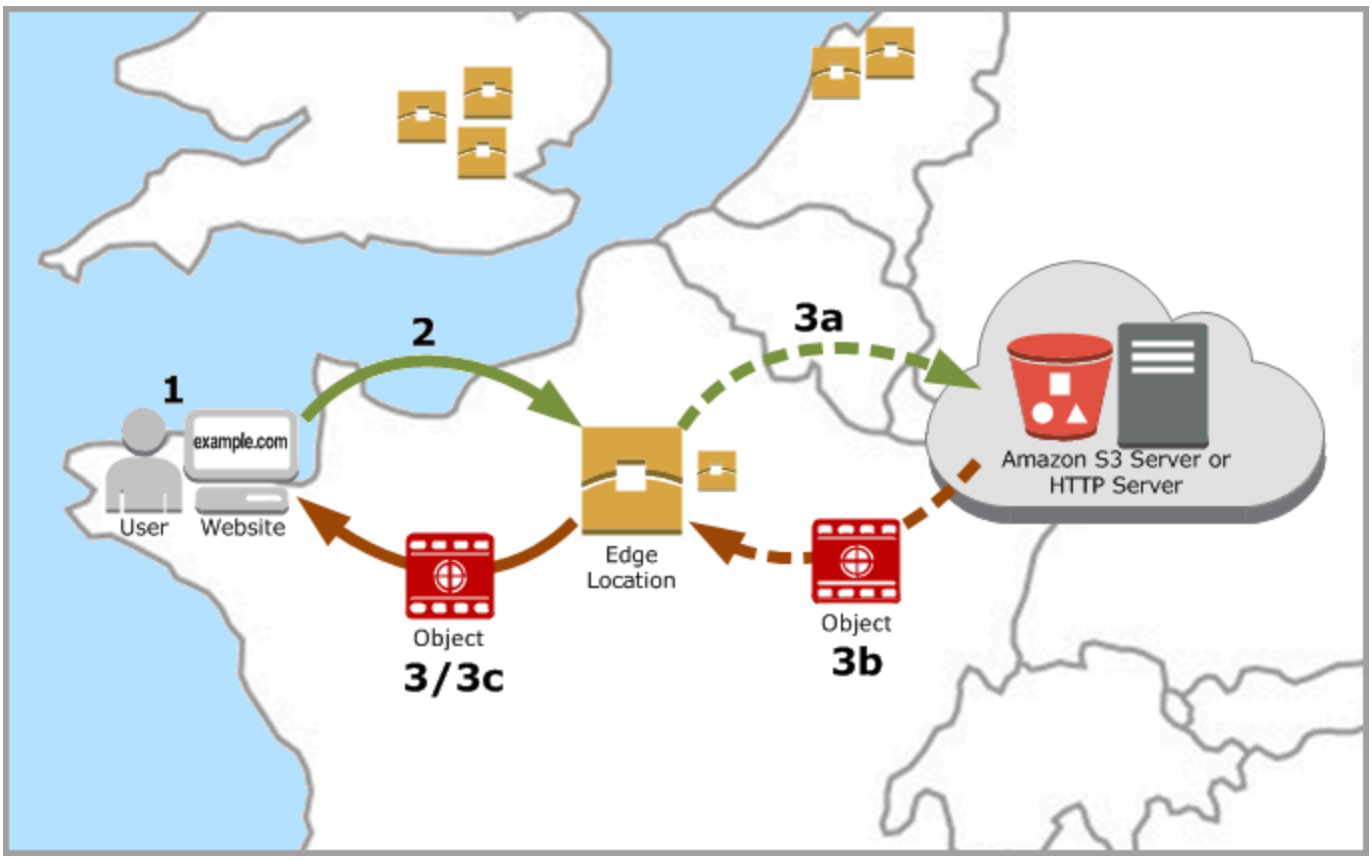
- How CloudFront delivers content to your users
- How CloudFront works with regional edge caches

How CloudFront delivers content to your users

After you configure CloudFront to deliver your content, here’s what happens when users request your objects:

1. A user accesses your website or application and sends a request for an object, such as an image file or an HTML file.

2. DNS routes the request to the CloudFront POP (edge location) that can best serve the request, typically the nearest CloudFront POP in terms of latency.
3. CloudFront checks its cache for the requested object. If the object is in the cache, CloudFront returns it to the user. If the object is *not* in the cache, CloudFront does the following:
   a. CloudFront compares the request with the specifications in your distribution and forwards the request to your origin server for the corresponding object—for example, to your Amazon S3 bucket or your HTTP server.
   b. The origin server sends the object back to the edge location.
   c. As soon as the first byte arrives from the origin, CloudFront begins to forward the object to the user. CloudFront also adds the object to the cache for the next time someone requests it.



How CloudFront works with regional edge caches

CloudFront points of presence (also known as *POPs* or *edge locations*) make sure that popular content can be served quickly to your viewers. CloudFront also has *regional edge caches* that bring more of your content closer to your viewers, even when the content is not popular enough to stay at a POP, to help improve performance for that content.

Regional edge caches help with all types of content, particularly content that tends to become less popular over time. Examples include user-generated content, such as video, photos, or artwork; e-commerce assets such as product photos and videos; and news and event-related content that might suddenly find new popularity.

How regional caches work

Regional edge caches are CloudFront locations that are deployed globally, close to your viewers. They're located between your origin server and the POPs—global edge locations that serve content directly to viewers. As objects become less popular, individual POPs might remove those objects to make room for more popular content. Regional edge caches have a larger cache than an individual POP, so objects remain in the cache longer at the nearest regional edge cache location. This helps keep more of your content closer to your viewers, reducing the need for CloudFront to go back to your origin server, and improving overall performance for viewers.

When a viewer makes a request on your website or through your application, DNS routes the request to the POP that can best serve the user's request. This location is typically the nearest CloudFront edge location in terms of latency. In the POP, CloudFront checks its cache for the requested object. If the object is in the cache, CloudFront returns it to the user. If the object is not in the cache, the POP typically goes to the nearest regional edge cache to fetch it. For more information about when the POP skips the regional edge cache and goes directly to the origin, see the following note.

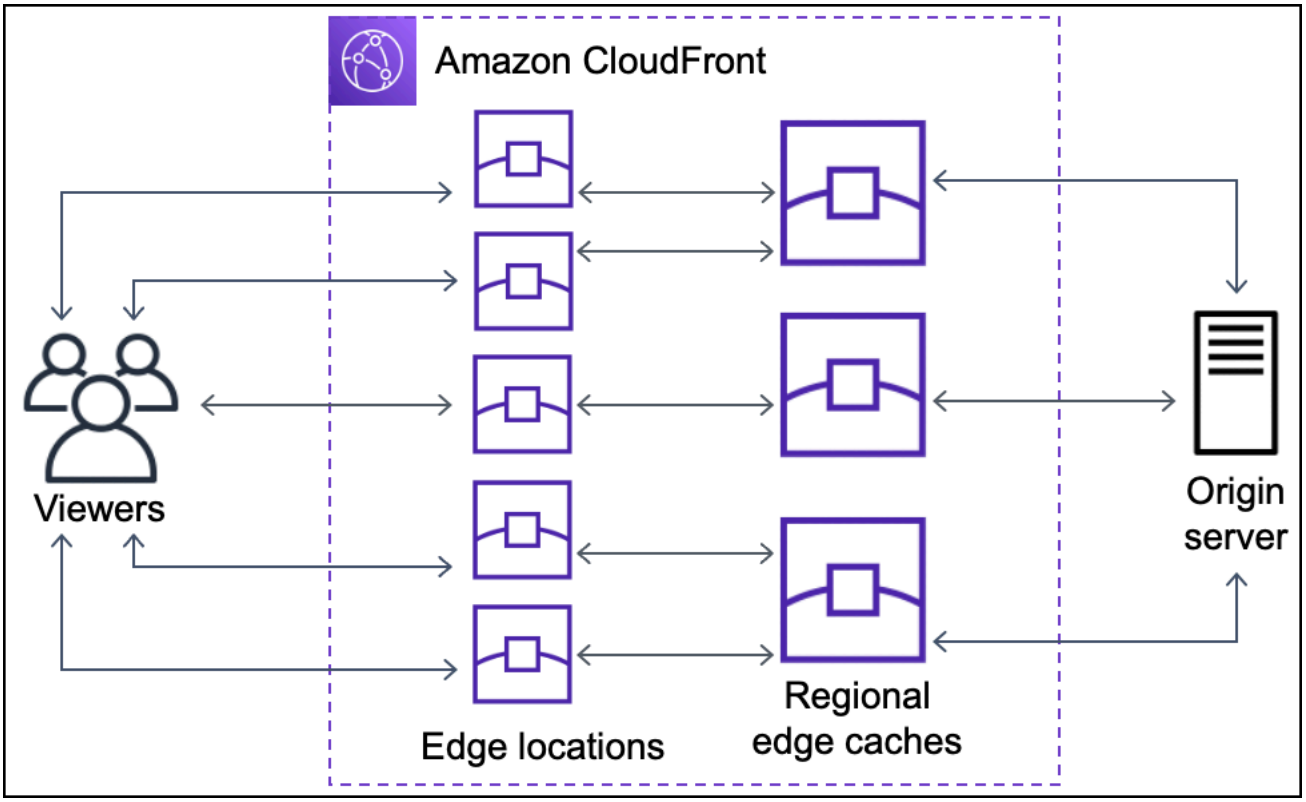
In the regional edge cache location, CloudFront again checks its cache for the requested object. If the object is in the cache, CloudFront forwards it to the POP that requested it. As soon as the first byte arrives from regional edge cache location, CloudFront begins to forward the object to the user. CloudFront also adds the object to the cache in the POP for the next time someone requests it.

For objects not cached at either the POP or the regional edge cache location, CloudFront compares the request with the specifications in your distributions and forwards the request to the origin server. After your origin server sends the object back to the regional edge cache location, it is forwarded to the POP, and then CloudFront forwards it to the user. In this case, CloudFront also adds the object to the cache in the regional edge cache location in addition to the POP for the next time a viewer requests it. This makes sure that all of the POPs in a region share a local cache, eliminating multiple requests to origin servers. CloudFront also keeps persistent connections with origin servers so objects are fetched from the origins as quickly as possible.

Note

- Regional edge caches have feature parity with POPs. For example, a cache invalidation request removes an object from both POP caches and regional edge caches before it expires. The next time a viewer requests the object, CloudFront returns to the origin to fetch the latest version of the object.
- Proxy HTTP methods (`PUT`, `POST`, `PATCH`, `OPTIONS`, and `DELETE`) go directly to the origin from the POPs and do not proxy through the regional edge caches.
- Dynamic requests, as determined at request time, do not flow through regional edge caches, but go directly to the origin.
- When the origin is an Amazon S3 bucket and the request's optimal regional edge cache is in the same AWS Region as the S3 bucket, the POP skips the regional edge cache and goes directly to the S3 bucket.

The following diagram illustrates how requests and responses flow through CloudFront edge locations and regional edge caches.



Locations and IP address ranges of CloudFront edge servers

For a list of the locations of CloudFront edge servers, see the Amazon CloudFront Global Edge Network page.

Amazon Web Services (AWS) publishes its current IP address ranges in JSON format. To view the current ranges, download ip-ranges.json. For more information, see AWS IP address ranges in the *Amazon Web Services General Reference*.

To find the IP address ranges that are associated with CloudFront edge servers, search ip-ranges.json for the following string:

```
"region": "GLOBAL",
"service": "CLOUDFRONT"
```

Alternatively, you can view only the CloudFront IP ranges at https://d7uri8nf7uskq.cloudfront.net/tools/list-cloudfront-ips.

Use the CloudFront managed prefix list

The CloudFront managed prefix list contains the IP address ranges of all of the CloudFront globally distributed origin-facing servers. If your origin is hosted on AWS and protected by an Amazon VPC security group, you can use the CloudFront managed prefix list. The prefix list allows inbound traffic to your origin only from CloudFront origin-facing servers, which prevents non-CloudFront traffic from reaching your origin. CloudFront maintains the managed prefix list so it's always up to date with the IP addresses of all of the CloudFront global origin-facing servers. With the CloudFront managed prefix list, you don't need to read or maintain a list of IP address ranges yourself.

For example, imagine that your origin is an Amazon EC2 instance in the Europe (London) Region (`eu-west-2`). If the instance is in a VPC, you can create a security group rule that allows inbound HTTPS access from the CloudFront managed prefix list. This allows all of the CloudFront global origin-facing servers to reach the instance. If you remove all other inbound rules from the security group, you prevent any non-CloudFront traffic from reaching the instance.

The CloudFront managed prefix lists are as follows:

- `com.amazonaws.global.cloudfront.origin-facing` (IPv4)
- `com.amazonaws.global.ipv6.cloudfront.origin-facing` (IPv6)

For more information, see Use an AWS-managed prefix list in the *Amazon VPC User Guide*.

> **Important**
>
> The CloudFront managed prefix list is unique in how it applies to Amazon VPC quotas. For more information, see AWS-managed prefix list weight in the *Amazon VPC User Guide*.

Using CloudFront with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

CloudFront technical resources

Use the following resources to get answers to technical questions about CloudFront:

- AWS re:Post – A community-based question and answer site for developers to discuss technical questions related to CloudFront.
- Support Center – This site includes information about your recent support cases and results from AWS Trusted Advisor and health checks. It also provides links to discussion forums, technical FAQs, the service health dashboard, and information about Support plans.
- AWS Premium Support – Learn about AWS Premium Support, a one-on-one, fast-response support channel that helps you build and run applications on AWS.

Get started with CloudFront

The topics in this section show you how to get started delivering your content with Amazon CloudFront.

The Set up your AWS account topic describes prerequisites for the following tutorials, such as creating an AWS account and creating a user with administrative access.

The basic distribution tutorial shows you how to set up origin access control (OAC) to send authenticated requests to an Amazon S3 origin.

The secure static website tutorial shows you how to create a secure static website for your domain name using OAC with an Amazon S3 origin. The tutorial uses an Amazon CloudFront (CloudFront) template for configuration and deployment.

The AWS CLI tutorial shows you how to create, update, and delete a standard distribution by using the AWS Command Line Interface (AWS CLI). For more information, see Get started with a standard distribution (AWS CLI).

Topics

- Set up your AWS account
- Get started with a CloudFront standard distribution
- Get started with a standard distribution (AWS CLI)
- Get started with a secure static website

Set up your AWS account

This topic describes preliminary steps, such as creating an AWS account, to prepare you to use Amazon CloudFront.

Topics

- Sign up for an AWS account
- Choose how to access CloudFront

Sign up for an AWS account

To get started with AWS, you need an AWS account. There are two ways to sign up for AWS. If you want complete control over account configuration and where your content is stored, or if you have regulated workloads such as HIPAA or FedRAMP, use Sign up for AWS (advanced). If you are a new user and want to use a login you already own to sign in to AWS and create an AWS environment with preconfigured defaults in select AWS Regions, use Sign up for AWS.

Choose how to access CloudFront

You can access Amazon CloudFront in the following ways:

- **AWS Management Console** – The procedures throughout this guide explain how to use the AWS Management Console to perform tasks.
- **AWS SDKs** – If you're using a programming language that AWS provides an SDK for, you can use an SDK to access CloudFront. SDKs simplify authentication, integrate easily with your development environment, and provide access to CloudFront commands. For more information, see Using CloudFront with an AWS SDK.
- **CloudFront API** – If you're using a programming language that an SDK isn't available for, see the Amazon CloudFront API Reference for information about API actions and about how to make API requests.
- **AWS CLI** – The AWS Command Line Interface (AWS CLI) is a unified tool for managing AWS services. For information about how to install and configure the AWS CLI, see Install or update to the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.
- **Tools for Windows PowerShell** – If you have experience with Windows PowerShell, you might prefer to use AWS Tools for Windows PowerShell. For more information, see Installing the AWS Tools for Windows PowerShell in the *AWS Tools for PowerShell User Guide*.

Get started with a CloudFront standard distribution

The procedures in this section show you how to use CloudFront to set up a standard distribution that does the following:

- Creates an S3 bucket to use as your distribution origin.
- Stores the original versions of your objects in an Amazon Simple Storage Service (Amazon S3) bucket.

- Uses origin access control (OAC) to send authenticated requests to your Amazon S3 origin. OAC sends requests through CloudFront to prevent viewers from accessing your S3 bucket directly. For more information about OAC, see Restrict access to an Amazon S3 origin.
- Uses the CloudFront domain name in URLs for your objects (for example, `https://d111111abcdef8.cloudfront.net/index.html`).
- Keeps your objects in CloudFront edge locations for the default duration of 24 hours (the minimum duration is 0 seconds).

Most of this is configured automatically for you when you create a CloudFront distribution.

Topics

- Prerequisites
- Create an Amazon S3 bucket
- Upload the content to the bucket
- Create a CloudFront distribution that uses an Amazon S3 origin with OAC
- Access your content through CloudFront
- Clean up
- Enhance your basic distribution

Prerequisites

Before you begin, make sure that you've completed the steps in Set up your AWS account.

Create an Amazon S3 bucket

An Amazon S3 bucket is a container for files (objects) or folders. CloudFront can distribute almost any type of file for you when an S3 bucket is the source. For example, CloudFront can distribute text, images, and videos. There is no maximum for the amount of data that you can store on Amazon S3.

For this tutorial, you create an S3 bucket with the provided sample `hello world` files that you will use to create a basic webpage.

To create a bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. We recommend that you use our Hello World sample for this Getting started. Download the *hello world* webpage: hello-world-html.zip. Unzip it and save the `css` folder and `index` file in a convenient location, such as the desktop where you are running your browser.
3. Choose **Create bucket**.
4. Enter a unique **Bucket name** that conforms to the General purpose buckets naming rules in the *Amazon Simple Storage Service User Guide*.
5. For **Region**, we recommend choosing an AWS Region that is geographically close to you. (This reduces latency and costs.)
   - Choosing a different Region works, too. You might do this to address regulatory requirements, for example.
6. Leave all other settings at their defaults, and then choose **Create bucket**.

Upload the content to the bucket

After you create your Amazon S3 bucket, upload the contents of the unzipped `hello world` file to it. (You downloaded and unzipped this file in Create an Amazon S3 bucket.)

To upload the content to Amazon S3

1. In the **General purpose buckets** section, choose the name of your new bucket.
2. Choose **Upload**.
3. On the **Upload** page, drag the `css` folder and `index` file into the drop area.
4. Leave all other settings at their defaults, and then choose **Upload**.

Create a CloudFront distribution that uses an Amazon S3 origin with OAC

For this tutorial, you will create a CloudFront distribution that uses an Amazon S3 origin with origin access control (OAC). OAC helps you securely send authenticated requests to your Amazon S3 origin. For more information about OAC, see Restrict access to an Amazon S3 origin.

To create a CloudFront distribution with an Amazon S3 origin that uses OAC

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **Create distribution**.
3. Enter a **Distribution name** for the standard distribution. The name will appear as the value for the `Name` key as a tag. You can change this value later. You can add up to 50 tags for your standard distribution. For more information, see Tag a CloudFront resource.
4. Choose **Single website or app**, **Next**.
5. Choose **Next**.
6. For **Origin type** page, select the **Amazon S3**.
7. For **S3 origin**, choose **Browse S3** and select the S3 bucket that you created for this tutorial.
8. For **Settings**, choose **Use recommended origin settings**. CloudFront will use the default recommended cache and origin settings for your Amazon S3 origin, including setting up Origin Access Control (OAC). For more information about the recommended settings, see Preconfigured distribution settings reference.
9. Choose **Next**.
10. On the **Enable security protections** page, choose whether to enable AWS WAF security protections.
11. Choose **Next**.
12. Choose **Create distribution**. CloudFront updates the S3 bucket policy for you.
13. Review the **Details** section for your new distribution. When your distribution is done deploying, the **Last modified** field changes from **Deploying** to a date and time.
14. Record the domain name that CloudFront assigns to your distribution. It looks similar to the following: `d111111abcdef8.cloudfront.net`.

Before using the distribution and S3 bucket from this tutorial in a production environment, make sure to configure it to meet your specific needs. For information about configuring access in a production environment, see *Configure secure access and restrict access to content*.

Access your content through CloudFront

To access your content through CloudFront, combine the domain name for your CloudFront distribution with the main page for your content. (You recorded your distribution domain name in Create a CloudFront distribution that uses an Amazon S3 origin with OAC.)

- Your distribution domain name might look like this: `d111111abcdef8.cloudfront.net`.
- The path to the main page of a website is typically `/index.html`.

Therefore, the URL to access your content through CloudFront might look like this:

`https://d111111abcdef8.cloudfront.net/index.html`.

If you followed the previous steps and used the *hello world* webpage, you should see a webpage that says **Hello world!**.

When you upload more content to this S3 bucket, you can access the content through CloudFront by combining the CloudFront distribution domain name with the path to the object in the S3 bucket. For example, if you upload a new file named `new-page.html` to the root of your S3 bucket, the URL looks like this:

`https://d111111abcdef8.cloudfront.net/new-page.html`.

Clean up

If you created your distribution and S3 bucket only as a learning exercise, delete them so that you no longer accrue charges. Delete the distribution first. For more information, see the following links:

- Delete a distribution
- Deleting a bucket

Enhance your basic distribution

This Get started tutorial provides a minimal framework for creating a distribution. We recommend that you explore the following enhancements:

- You can use the CloudFront private content feature to restrict access to the content in the Amazon S3 buckets. For more information about distributing private content, see Serve private content with signed URLs and signed cookies.
- You can configure your CloudFront distribution to use a custom domain name (for example, `www.example.com` instead of `d111111abcdef8.cloudfront.net`). For more information, see Use custom URLs.
- This tutorial uses an Amazon S3 origin with origin access control (OAC). However, you can't use OAC if your origin is an S3 bucket configured as a website endpoint. If that's the case, you must set up your bucket with CloudFront as a custom origin. For more information, see Use an Amazon S3 bucket that's configured as a website endpoint. For more information about OAC, see Restrict access to an Amazon S3 origin.

Get started with a standard distribution (AWS CLI)

The procedures in this section show you how to use the AWS CLI with CloudFront to set up a basic configuration that involves the following:

- Creating an Amazon S3 bucket to use as your distribution origin.
- Storing the original versions of your objects in the S3 bucket.
- Using origin access control (OAC) to send authenticated requests to your Amazon S3 origin. OAC sends requests through CloudFront to prevent viewers from accessing your S3 bucket directly. For more information about OAC, see Restrict access to an Amazon S3 origin.
- Using the CloudFront domain name in URLs for your objects (for example, `https://d111111abcdef8.cloudfront.net/index.html`).
- Keeping your objects in CloudFront edge locations for the default duration of 24 hours (the minimum duration is 0 seconds).

Most of these options are customizable. For information about how to customize your CloudFront distribution options, see Create a distribution.

Prerequisites

Before you begin, make sure that you've completed the steps in Set up your AWS account.

Install the AWS CLI and configure it with your credentials. For more information, see Getting started with the AWS CLI in the *AWS CLI User Guide*.

Create an Amazon S3 bucket

An Amazon S3 bucket is a container for files (objects) or folders. CloudFront can distribute almost any type of file for you when an S3 bucket is the source. For example, CloudFront can distribute text, images, and videos. There is no maximum for the amount of data that you can store on Amazon S3.

For this tutorial, you create an S3 bucket and upload an HTML file that you will use to create a basic webpage.

```
aws s3 mb s3://amzn-s3-demo-bucket/ --region us-east-1
```

Replace *amzn-s3-demo-bucket* with a globally unique bucket name. For the AWS Region, we recommend choosing a Region that is geographically close to you. This reduces latency and costs, but choosing a different Region works, too. For example, you might do this to address regulatory requirements.

Upload the content to the bucket

For this tutorial, download and extract the sample content files for a basic "Hello World" webpage.

```
# Create a temporary directory
mkdir -p ~/cloudfront-demo

# Download the sample Hello World files
curl -o ~/cloudfront-demo/hello-world-html.zip https://docs.aws.amazon.com/
AmazonCloudFront/latest/DeveloperGuide/samples/hello-world-html.zip

# Extract the zip file
unzip ~/cloudfront-demo/hello-world-html.zip -d ~/cloudfront-demo/hello-world
```

This creates a directory with an `index.html` file and a `css` folder. Upload these files to your S3 bucket.

```
aws s3 cp ~/cloudfront-demo/hello-world/ s3://amzn-s3-demo-bucket/ --recursive
```

Create an Origin Access Control (OAC)

For this tutorial, you will create an origin access control (OAC). OAC helps you securely send authenticated requests to your Amazon S3 origin. For more information about OAC, see Restrict access to an Amazon S3 origin.

```
aws cloudfront create-origin-access-control \
    --origin-access-control-config Name="oac-for-
s3",SigningProtocol=sigv4,SigningBehavior=always,OriginAccessControlOriginType=s3
```

Save the OAC ID from the output as an environment variable. Replace the example value with your own OAC ID. You will use this in the next step.

```
OAC_ID="E1ABCD2EFGHIJ"
```

Create a standard distribution

Create a distribution configuration file named `distribution-config.json`. Replace the example bucket name with your bucket name for the `Id`, `DomainName`, and `TargetOriginId` values.

```
cat > distribution-config.json << EOF
{
    "CallerReference": "cli-example-$(date +%s)",
    "Origins": {
        "Quantity": 1,
        "Items": [
            {
                "Id": "S3-amzn-s3-demo-bucket",
                "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                "S3OriginConfig": {
                    "OriginAccessIdentity": ""
                },
                "OriginAccessControlId": "$OAC_ID"
            }
        ]
    },
    "DefaultCacheBehavior": {
        "TargetOriginId": "S3-amzn-s3-demo-bucket",
        "ViewerProtocolPolicy": "redirect-to-https",
        "AllowedMethods": {
```

```
            "Quantity": 2,
            "Items": ["GET", "HEAD"],
            "CachedMethods": {
                "Quantity": 2,
                "Items": ["GET", "HEAD"]
            }
        },
        "DefaultTTL": 86400,
        "MinTTL": 0,
        "MaxTTL": 31536000,
        "Compress": true,
        "ForwardedValues": {
            "QueryString": false,
            "Cookies": {
                "Forward": "none"
            }
        }
    },
    "Comment": "CloudFront distribution for S3 bucket",
    "Enabled": true
}
EOF
```

Create the standard distribution.

```
aws cloudfront create-distribution --distribution-config file://distribution-
config.json
```

Save the distribution ID and domain name from the output as environment variables. Replace the example values with your own. You'll use these later in this tutorial.

```
DISTRIBUTION_ID="EABCD1234XMPL"
DOMAIN_NAME="d111111abcdef8.cloudfront.net"
```

Before using the distribution and S3 bucket from this tutorial in a production environment, make sure to configure it to meet your specific needs. For information about configuring access in a production environment, see *Configure secure access and restrict access to content*.

Update your S3 bucket policy

Update your S3 bucket policy to allow CloudFront to access the objects. Replace the example bucket name with your bucket name.

```
# Get your AWS account ID
ACCOUNT_ID=$(aws sts get-caller-identity --query 'Account' --output text)

# Create the bucket policy
cat > bucket-policy.json << EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontServicePrincipal",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceArn": "arn:aws:cloudfront::$ACCOUNT_ID:distribution/
$DISTRIBUTION_ID"
                }
            }
        }
    ]
}
EOF

# Apply the bucket policy
aws s3api put-bucket-policy \
    --bucket amzn-s3-demo-bucket \
    --policy file://bucket-policy.json
```

Confirm distribution deployment

After you create your distribution, it will take some time to finish deploying. When the distribution status changes from `InProgress` to `Deployed`, proceed to the next step.

```
aws cloudfront get-distribution --id $DISTRIBUTION_ID --query 'Distribution.Status'
```

Alternatively, you can use the `wait` command to wait for distribution deployment.

```
aws cloudfront wait distribution-deployed --id $DISTRIBUTION_ID
```

Access your content through CloudFront

To access your content through CloudFront, combine the domain name for your CloudFront distribution with the main page for your content. Replace the example CloudFront domain name with your own.

```
https://d111111abcdef8.cloudfront.net/index.html
```

If you followed the previous steps and created the HTML file, you should see a webpage that says **Hello world!**.

When you upload more content to this S3 bucket, you can access the content through CloudFront by combining the CloudFront distribution domain name with the path to the object in the S3 bucket. For example, if you upload a new file named `new-page.html` to the root of your S3 bucket, the URL looks like this:

`https://d111111abcdef8.cloudfront.net/new-page.html`.

Clean up

If you created your distribution and S3 bucket only as a learning exercise, delete them so that you no longer accrue charges. Disable and delete the distribution first.

To disable and delete a standard distribution (AWS CLI)

1. First, disable the distribution.

```
# Get the current configuration and ETag
ETAG=$(aws cloudfront get-distribution-config --id $DISTRIBUTION_ID --query 'ETag'
 --output text)

# Create a modified configuration with Enabled=false
aws cloudfront get-distribution-config --id $DISTRIBUTION_ID | \
jq '.DistributionConfig.Enabled = false' > temp_disabled_config.json

# Update the distribution to disable it
aws cloudfront update-distribution \
    --id $DISTRIBUTION_ID \
```

```
    --distribution-config file://<(jq '.DistributionConfig'
 temp_disabled_config.json) \
    --if-match $ETAG
```

2. Wait for the distribution to be disabled.

```
aws cloudfront wait distribution-deployed --id $DISTRIBUTION_ID
```

3. Delete the distribution.

```
# Get the current ETag
ETAG=$(aws cloudfront get-distribution-config --id $DISTRIBUTION_ID --query 'ETag'
 --output text)

# Delete the distribution
aws cloudfront delete-distribution --id $DISTRIBUTION_ID --if-match $ETAG
```

To delete an S3 bucket (AWS CLI)

- Delete the S3 bucket and its contents. Replace the example bucket name with your own.

```
# Delete the bucket contents
aws s3 rm s3://amzn-s3-demo-bucket --recursive

# Delete the bucket
aws s3 rb s3://amzn-s3-demo-bucket
```

To clean up the local files created for this tutorial, run the following commands:

```
# Clean up local files
rm -f distribution-config.json bucket-policy.json temp_disabled_config.json
rm -rf ~/cloudfront-demo
```

Optionally, you can delete the OAC that you created for this tutorial.

```
# Get the OAC ETag
OAC_ETAG=$(aws cloudfront get-origin-access-control --id $OAC_ID --query 'ETag' --
output text)

# Delete the OAC
```

```
aws cloudfront delete-origin-access-control --id $OAC_ID --if-match $OAC_ETAG
```

Get started with a secure static website

You can get started with Amazon CloudFront by using the solution described in this topic to create a secure static website for your domain name. A *static website* uses only static files—like HTML, CSS, JavaScript, images, and videos—and doesn't need servers or server-side processing. With this solution, your website gets the following benefits:

- **Uses the durable storage of Amazon Simple Storage Service (Amazon S3)** – This solution creates an Amazon S3 bucket to host your static website's content. To update your website, just upload your new files to the S3 bucket.
- **Is sped up by the Amazon CloudFront content delivery network** – This solution creates a CloudFront distribution to serve your website to viewers with low latency. The distribution is configured with origin access control (OAC) to make sure that the website is accessible only through CloudFront, not directly from S3.
- **Is secured by HTTPS and security headers** – This solution creates an SSL/TLS certificate in AWS Certificate Manager (ACM), and attaches it to the CloudFront distribution. This certificate enables the distribution to serve your domain's website securely with HTTPS.
- **Is configured and deployed with AWS CloudFormation** – This solution uses a CloudFormation template to set up all the components, so you can focus more on your website's content and less on configuring components.

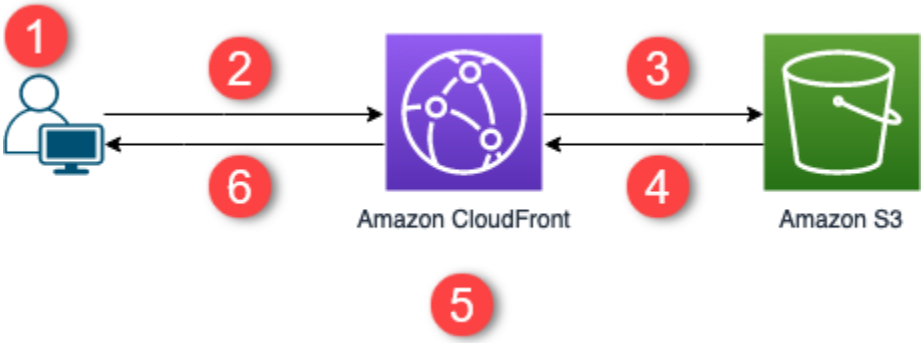
This solution is open source on GitHub. To view the code, submit a pull request, or open an issue, go to https://github.com/aws-samples/amazon-cloudfront-secure-static-site.

Topics

- Solution overview
- Deploy the solution

Solution overview

The following diagram shows an overview of how this static website solution works:



1. The viewer requests the website at www.example.com.
2. If the requested object is cached, CloudFront returns the object from its cache to the viewer.
3. If the object is not in the CloudFront cache, CloudFront requests the object from the origin (an S3 bucket).
4. S3 returns the object to CloudFront.
5. CloudFront caches the object.
6. The objects is returned to the viewer. Subsequent requests for the object that come to the same CloudFront edge location are served from the CloudFront cache.

Deploy the solution

To deploy this secure static website solution, you can choose from either of the following options:

- Use the CloudFormation console to deploy the solution with default content, then upload your website content to Amazon S3.
- Clone the solution to your computer to add your website content. Then, deploy the solution with the AWS Command Line Interface (AWS CLI).

> **Note**
>
> You must use the US East (N. Virginia) Region to deploy the CloudFormation template.

Topics

- Prerequisites
- Use the CloudFormation console

- Clone the solution locally
- Finding access logs

Prerequisites

To use this solution, you must have the following prerequisites:

- A registered domain name, such as example.com, that's pointed to an Amazon Route 53 hosted zone. The hosted zone must be in the same AWS account where you deploy this solution. If you don't have a registered domain name, you can register one with Route 53. If you have a registered domain name but it's not pointed to a Route 53 hosted zone, configure Route 53 as your DNS service.
- AWS Identity and Access Management (IAM) permissions to launch CloudFormation templates that create IAM roles, and permissions to create all the AWS resources in the solution. For more information, see Controlling access with AWS Identity and Access Management in the *AWS CloudFormation User Guide*.

You are responsible for the costs incurred while using this solution. For more information about costs, see the pricing pages for each AWS service.

Use the CloudFormation console

To deploy using the CloudFormation console

1. Launch this solution in the CloudFormation console. If necessary, sign in to your AWS account.
2. The **Create stack** wizard opens in the CloudFormation console, with prepopulated fields that specify this solution's CloudFormation template.

   At the bottom of the page, choose **Next**.
3. On the **Specify stack details** page, enter values for the following fields:

   - **SubDomain** – Enter the subdomain to use for your website. For example, if the subdomain is *www*, your website is available at *`www.example.com.`* (Replace example.com with your domain name, as explained in the following bullet.)
   - **DomainName** – Enter your domain name, such as *`example.com`*. This domain must be pointed to a Route 53 hosted zone.
   - **HostedZoneId** – The Route 53 hosted zone ID of your domain name.

- **CreateApex** – (Optional) Create an alias to the domain apex (example.com) in your CloudFront configuration.

4. When finished, choose **Next**.
5. (Optional) On the **Configure stack options** page, add tags and other stack options.
6. When finished, choose **Next**.
7. On the **Review** page, scroll to the bottom of the page, then select the two boxes in the **Capabilities** section. These capabilities allow CloudFormation to create an IAM role that allows access to the stack's resources, and to name the resources dynamically.
8. Choose **Create stack**.
9. Wait for the stack to finish creating. The stack creates some nested stacks, and can take several minutes to finish. When it's finished, the **Status** changes to **CREATE_COMPLETE**.

   When the status is **CREATE_COMPLETE**, go to https://*www.example.com* to view your website (replace www.example.com with the subdomain and domain name that you specified in step 3). You should see the website's default content:



To replace the website's default content with your own

1. Open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. Choose the bucket whose name begins with **amazon-cloudfront-secure-static-site-s3bucketroot-**.

   > **Note**
   >
   > Make sure to choose the bucket with **s3bucketroot** in its name, not **s3bucketlogs**. The bucket with **s3bucketroot** in its name contains the website content. The one with **s3bucketlogs** contains only log files.

3. Delete the website's default content, then upload your own.

> **Note**
>
> If you viewed your website with this solution's default content, then it's likely that some of the default content is cached in a CloudFront edge location. To make sure that viewers see your updated website content, *invalidate* the files to remove the cached copies from CloudFront edge locations. For more information, see Invalidate files to remove content.

Clone the solution locally

Prerequisites

To add your website content before deploying this solution, you must package the solution's artifacts locally, which requires Node.js and npm. For more information, see https://www.npmjs.com/get-npm.

To add your website content and deploy the solution

1. Clone or download the solution from https://github.com/aws-samples/amazon-cloudfront-secure-static-site. After you clone or download it, open a command prompt or terminal and navigate to the `amazon-cloudfront-secure-static-site` folder.
2. Run the following command to install and package the solution's artifacts:

```
make package-static
```

3. Copy your website's content into the `www` folder, overwriting the default website content.
4. Run the following AWS CLI command to create an Amazon S3 bucket to store the solution's artifacts. Replace *amzn-s3-demo-bucket-for-artifacts* with your own bucket name.

```
aws s3 mb s3://amzn-s3-demo-bucket-for-artifacts --region us-east-1
```

5. Run the following AWS CLI command to package the solution's artifacts as a CloudFormation template. Replace *amzn-s3-demo-bucket-for-artifacts* with the name of the bucket that you created in the previous step.

```
aws cloudformation package \
    --region us-east-1 \
```

```
    --template-file templates/main.yaml \
    --s3-bucket amzn-s3-demo-bucket-for-artifacts \
    --output-template-file packaged.template
```

6. Run the following command to deploy the solution with CloudFormation, replacing the following values:

   - *your-CloudFormation-stack-name* – Replace with a name for the CloudFormation stack.
   - *example.com* – Replace with your domain name. This domain must be pointed to a Route 53 hosted zone in the same AWS account.
   - *www* – Replace with the subdomain to use for your website. For example, if the subdomain is *www*, your website is available at www.example.com.
   - *hosted-zone-ID* – Replace with the Route 53 hosted zone ID of your domain name.

   ```
   aws cloudformation deploy \
       --region us-east-1 \
       --stack-name your-CloudFormation-stack-name \
       --template-file packaged.template \
       --capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND \
       --parameter-overrides DomainName=example.com SubDomain=www HostedZoneId=hosted-
   zone-ID
   ```

   - (Optional) To deploy the stack with a domain apex, run the following command instead.

     ```
     aws --region us-east-1 cloudformation deploy \
         --stack-name your-CloudFormation-stack-name \
         --template-file packaged.template \
         --capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND \
         --parameter-overrides  DomainName=example.com SubDomain=www
      HostedZoneId=hosted-zone-ID CreateApex=yes
     ```

7. Wait for the CloudFormation stack to finish creating. The stack creates some nested stacks, and can take several minutes to finish. When it's finished, the **Status** changes to **CREATE_COMPLETE**.

   When the status changes to **CREATE_COMPLETE**, go to https://www.example.com to view your website (replace www.example.com with the subdomain and domain name that you specified in the previous step). You should see your website's content.

Finding access logs

This solution enables access logs for the CloudFront distribution. Complete the following steps to locate the distribution's access logs.

To locate the distribution's access logs

1. Open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. Choose the bucket whose name begins with **amazon-cloudfront-secure-static-site-s3bucketlogs-**.

> **Note**
>
> Make sure to choose the bucket with **s3bucketlogs** in its name, not **s3bucketroot**. The bucket with **s3bucketlogs** in its name contains log files. The one with **s3bucketroot** contains the website content.

3. The folder named **cdn** contains the CloudFront access logs.

CloudFront for e-commerce and marketplace applications

An e-commerce or marketplace application serves three distinct content types through a single CloudFront distribution: static assets (product images, CSS, JavaScript), dynamic pages (product listings, search results), and API responses (cart, inventory, pricing). Each content type requires different cache behavior, TTL, and origin configuration to balance freshness against performance.

For the security controls that apply to marketplace and e-commerce distributions, see Security checklist for marketplace distributions.

Architecture overview

A marketplace CloudFront distribution uses multiple cache behaviors to route requests to the correct origin based on URL path patterns. Static assets are served from Amazon Simple Storage Service (Amazon S3), while dynamic content and API requests are forwarded to an Elastic Load Balancing (Application Load Balancer) in front of your application servers. An ElastiCache (Valkey) cluster provides an application-level cache between the ALB and your database for frequently accessed product data.

Distribution architecture components

Component	Content served	Purpose
CloudFront distribution	All content (single domain)	Global edge caching, TLS terminati on, request routing by using cache behaviors
Amazon S3 origin (static assets)	Product images, CSS, JS, fonts	Durable object storage with origin access control (OAC). Immutable content with long TTLs.
ALB origin (dynamic content)	Product pages, search, API endpoints	Routes to application servers. Short TTLs or no caching for personalized content.

Component	Content served	Purpose
ElastiCache cluster (Valkey)	Product catalog, inventory counts, session data	Application-level cache between ALB and database. Sub-millisecond reads for frequently accessed data.

Cache behaviors for e-commerce

Cache behaviors determine how CloudFront handles requests based on URL path patterns. For a marketplace, configure separate behaviors for each content type with appropriate cache policies and TTLs.

Recommended cache behaviors

Path pattern	Origin	TTL	Cache policy	Rationale
`/static/*`	Amazon S3	365 days	CachingOptimized	Versioned filenames (for example, `app.a1b2c3.js`) allow maximum TTL. Cache invalidation is never needed — deploy new versions with new filenames.
`/images/*`	Amazon S3	30 days	CachingOptimized	Product images change infrequently. Use cache-tag invalidation when a seller updates an image. Lower TTL than static assets because image URLs might be reused.
`/api/products/*`	ALB	60 seconds	Custom (include query strings)	Product listing and search APIs change frequently but tolerate brief staleness . Include query string

Path pattern	Origin	TTL	Cache policy	Rationale
				parameters in the cache key for pagination and filters.
`/api/cart/*`	ALB	0 (no cache)	CachingDisabled	Cart operations are user-specific and must always reach the origin. Disable caching entirely.
`/api/inventory/*`	ALB	5 seconds	Custom (include query strings)	Inventory counts change rapidly during sales events. Very short TTL prevents overselling while reducing origin load.
`Default (*)`	ALB	0 (no cache)	CachingDisabled	Default behavior forwards uncached requests to the application for server-side rendering. Pages that include personalized content (recommendations, user name) should not be cached at the edge.

Note

Order cache behaviors from most specific to least specific. CloudFront evaluates path patterns in the order listed and uses the first match. Place `/api/cart/*` before `/api/*` to make sure cart requests bypass caching.

For more information about cache behaviors, see the section called “Cache behavior settings”.

TTL strategy for marketplace content

TTL configuration determines how long CloudFront serves cached content before checking the origin for updates. E-commerce applications require different TTL strategies depending on how content freshness affects the customer experience.

Immutable assets (CSS, JS, fonts)

Set the maximum TTL (365 days). Use content-hashed filenames so each deployment creates new URLs. Never invalidate — old versions expire naturally as users load the new filenames. Set `Cache-Control: public, max-age=31536000, immutable` at the origin.

Product images

Set TTL to 30 days with cache-tag invalidation on update. When a seller uploads a new image, invalidate by the product's cache tag rather than by path. This targets only the affected objects without broad wildcard invalidation. Set `Cache-Control: public, max-age=2592000` at the origin.

Product catalog API

Set TTL to 60 seconds. Product listings, descriptions, and pricing tolerate brief staleness. This reduces origin load during traffic spikes (flash sales, product launches) while keeping data reasonably current. Set `Cache-Control: public, max-age=60, stale-while-revalidate=30` to serve stale content while CloudFront fetches a fresh response in the background.

Inventory and pricing API

Set TTL to 5 seconds. These values change frequently and affect purchase decisions. Very short caching still reduces origin load by orders of magnitude during traffic spikes — thousands of concurrent users see the same 5-second cached response instead of each hitting the origin.

User-specific content (cart, recommendations)

Disable caching (TTL = 0). Content that varies per user must always reach the origin. If you cache user-specific content by mistake, CloudFront might serve one user's data to another user. Use the `CachingDisabled` managed policy for these behaviors.

Origin configuration

Amazon S3 origin for static assets

Use an Amazon S3 bucket as the origin for product images and static assets. Configure origin access control (OAC) to restrict direct access to the bucket — all requests must flow through CloudFront.

Amazon S3 origin settings

Setting	Value	Rationale
Origin domain	`marketplace-assets.s3.amazonaws.com`	Use the Amazon S3 REST API endpoint (not the website endpoint) to support OAC.
Origin access	Origin access control (OAC)	Restricts bucket access to CloudFront only. Prevents direct S3 URL access to your assets.
Origin Shield	Enabled (region closest to bucket)	Adds a centralized caching layer between edge locations and the origin. Reduces S3 request costs and improves cache hit ratio for images.
Connection attempts	3	Default. Retries on transient S3 errors.

ALB origin for dynamic content

Use an Application Load Balancer as the origin for dynamic product pages, search, and API endpoints. The ALB routes to your application servers (Amazon EC2 instances, Amazon ECS tasks, or Lambda functions by using target groups).

ALB origin settings

Setting	Value	Rationale
Origin domain	ALB DNS name	Use the ALB DNS name directly. Do not use an IP address — ALB IPs change.
Protocol	HTTPS only	Encrypts traffic between CloudFront and the origin. Required for sensitive data (user sessions, payment info).
Origin custom header	`X-Origin-Verify: <secret-value>`	Restricts ALB access to requests from CloudFront. The ALB checks for this header and rejects direct access attempts.
Connection timeout	10 seconds	Shorter than default (30s). Fail fast on origin issues rather than keeping edge connections waiting.
Response timeout	30 seconds	Allows time for complex search queries and catalog operations. Increase if your API has long-runn ing operations.
Keep-alive timeout	5 seconds	Reuses connections to the ALB. Reduces TLS handshake overhead for subsequent requests.

Application-level caching with ElastiCache (Valkey)

CloudFront caches content at the edge, but frequently accessed data that requires database queries benefits from an additional application-level cache. An ElastiCache cluster running Valkey sits between your application servers and the database, providing sub-millisecond reads for product catalog data, inventory counts, and session state.

Two-tier caching strategy

Layer	What it caches	TTL	Cache miss behavior
CloudFront edge	Full HTTP responses (API JSON, HTML pages, images)	5s – 365d (by behavior)	Forwards request to ALB origin
ElastiCache (Valkey)	Application data objects (product records, inventory, sessions)	30s – 5 min (by data type)	Application queries the database and writes the result to cache

On a product page request:

1. CloudFront checks its edge cache. On hit, returns the cached response immediately.
2. On miss, CloudFront forwards to the ALB.
3. The application checks ElastiCache for the product data. On hit, builds the response from cached data (sub-millisecond).
4. On ElastiCache miss, queries the database, writes to ElastiCache, and returns the response.
5. CloudFront caches the response at the edge according to the behavior's TTL setting.

When a seller updates a product:

1. The application invalidates the product key in ElastiCache.
2. The application sends a cache-tag invalidation to CloudFront for the product's tag.
3. The next request triggers a fresh response through both cache layers.

Frequently asked questions

The following sections answer common questions about caching strategies for e-commerce and marketplace workloads.

How do I handle personalized content with caching?

Separate personalized elements from cacheable content. Serve the page shell (product details, images, descriptions) from CloudFront cache, and load personalized elements (recommendations,

cart count, user name) by using client-side API calls that bypass caching. With this approach, you can cache the expensive page rendering and keep personalization current.

How do I choose between invalidation and short TTLs?

Use short TTLs (5–60 seconds) for content that changes frequently and predictably (inventory, pricing). Use invalidation for content that changes rarely but must update immediately when it does (product images after a seller edit, product descriptions after a compliance review). Invalidation has a per-request cost and a concurrency limit — do not use it as a substitute for appropriate TTLs.

How do I prepare for flash sales and traffic spikes?

CloudFront scales automatically to handle traffic spikes. To maximize cache hit ratio during a sale: pre-warm product pages by requesting them before the event starts, increase API TTLs temporarily (for example, increase inventory TTL from 5s to 15s) to absorb more traffic at the edge, and make sure your ElastiCache cluster has enough memory headroom for increased cache writes. Monitor the CloudFront cache hit ratio metric during the event.

Should I use Origin Shield for my marketplace?

Yes. Origin Shield adds a centralized cache layer between regional edge caches and your origin. For e-commerce workloads, it reduces origin requests for popular products (because a cache hit at Origin Shield serves all edge locations), reduces S3 GET costs for images, and smooths traffic spikes to the ALB. Enable it in the Region closest to your origin. For more information, see the section called “Use Origin Shield”.

Security checklist for marketplace distributions

E-commerce and marketplace applications handle sensitive data — customer sessions, payment flows, and premium content. This checklist covers the security controls specific to a CloudFront distribution with an ElastiCache (Valkey) caching layer. Apply these controls in addition to the general CloudFront security best practices described in *Configure secure access and restrict access to content*.

This checklist complements the configuration guidance in CloudFront for e-commerce and marketplace applications.

Signed URLs and cookies for premium content

Use signed URLs or signed cookies to restrict access to content that requires authorization — premium product previews, purchased digital goods, time-limited promotional content, or seller-specific assets.

Signed URL and signed cookie comparison

Method	Use when	E-commerce example
Signed URL	Restricting access to individua l files. Sharing download links. Clients that do not support cookies.	One-time download link for a purchased digital product. Time-limited access to a high-resolution product image for a verified seller.
Signed cookie	Providing access to multiple files (for example, all assets in a premium tier). Avoiding URL changes for existing content.	Premium membership access to an entire catalog of digital content. Subscriber-only product previews across multiple pages.

Configuration requirements:

- Create a key group with an RSA public key (2048-bit minimum). Store the private key in AWS Secrets Manager for your application to generate signatures.
- Set the cache behavior's **Restrict viewer access** to **Yes** and associate the key group.

- Set an expiration time appropriate to the content type: minutes for download links, hours for browsing sessions, days for subscription access.
- Use a custom policy (not a canned policy) when you need to restrict by IP range or allow wildcard paths.

For implementation details, see the section called “Restrict content with signed URLs and signed cookies”.

Origin access control (OAC) for Amazon S3

Origin access control prevents users from bypassing CloudFront to access your Amazon S3 bucket directly. Without OAC, anyone who discovers your bucket URL can access assets without going through your signed URL/cookie enforcement, WAF rules, or geo-restrictions. For full configuration steps, see the section called “Restrict access to an AWS origin”.

Enable OAC for all Amazon S3 origins

Create an OAC and associate it with each Amazon S3 origin in your distribution. Update the bucket policy to allow only the CloudFront service principal (`cloudfront.amazonaws.com`) with a condition matching your distribution ARN.

Remove public access

In the Amazon Simple Storage Service console, enable Amazon S3 Block Public Access on the bucket. This is an Amazon S3 setting, not a CloudFront setting. The bucket policy grants access only to CloudFront — no public ACLs or policies are needed. For more information, see Blocking public access to your Amazon S3 storage in the *Amazon Simple Storage Service User Guide*.

Use the Amazon S3 REST API endpoint

Configure the origin using the bucket's REST API endpoint (`bucket-name.s3.amazonaws.com`), not the Amazon S3 website endpoint. OAC only works with the REST API endpoint.

Encryption in transit

Encrypt all connections in the request path — viewer to CloudFront, CloudFront to origin, and application to ElastiCache cache.

Encryption in transit configuration

Connection	Setting	Configuration
Viewer → CloudFront	Viewer protocol policy: HTTPS only	Set viewer protocol policy to `redirect-to-https` or `https-only` on all cache behaviors. Use a TLS 1.2 minimum security policy (`TLSv1.2_2021`).
CloudFront → S3 origin	Origin protocol: HTTPS	S3 REST API endpoints support HTTPS by default. CloudFront uses HTTPS when you configure the origin with the REST API endpoint and OAC.
CloudFront → ALB origin	Origin protocol: HTTPS only	Set origin protocol policy to `https-only` . Install a valid TLS certificate on the ALB (use AWS Certificate Manager). CloudFront validates the certificate on connection.
Application → ElastiCache	In-transit encryption: Enabled	In the ElastiCache console, enable `TransitEncryptionEnabled` on the replication group. This is an ElastiCache setting. Use TLS connections from your application code. Valkey and Redis OSS 6.0 and later support in-transit encryption natively.

VPC security groups for the cache cluster

The ElastiCache cluster runs in a VPC and must only accept connections from your application servers. Use security group chaining to restrict access.

Security group configuration for ElastiCache

Security group	Inbound rule	Rationale
Cache cluster SG	TCP 6379 from Applicati on SG only	Only application servers can reach the cache. No internet access, no access from other services. Use port 6379 (default Valkey/Redis port) or your configured port.
Application SG	TCP 443 from ALB SG only	Application servers accept traffic only from the ALB. Combined with the cache cluster SG, this creates a chain: CloudFront → ALB → App → Cache.

Additional VPC hardening:

- Deploy the ElastiCache cluster in private subnets with no internet gateway route.
- Use a subnet group that spans at least 2 Availability Zones for high availability.
- Enable ElastiCache auth token (password) for an additional authentication layer beyond security groups.
- Disable public access to the ElastiCache cluster (default, but verify).

Additional security controls

Restrict ALB access to CloudFront only

Add a custom origin header (`X-Origin-Verify`) that CloudFront sends with every request. Configure the ALB or application to reject requests that do not include this header. Store the header value in AWS Secrets Manager and rotate it periodically. Alternatively, use the CloudFront managed prefix list to restrict ALB security group inbound rules to CloudFront IP ranges.

Enable AWS WAF on the distribution

Associate an AWS WAF web ACL with rate-limiting rules to protect against cart abuse, credential stuffing on login APIs, and inventory scraping. Use managed rule groups for common threats (SQL injection, XSS) and custom rules for application-specific patterns.

Geo-restrictions for compliance

If your marketplace operates in specific countries, enable geo-restriction to block requests from regions where you do not have legal authority to sell or ship. Apply at the distribution level. For per-path geo-restriction, use AWS WAF geo-match conditions in a web ACL rule with a URL path condition.

Access logging

Enable CloudFront standard logging or real-time logging to detect abnormal access patterns — sudden spikes in download link usage, access from unexpected regions, or repeated failed auth attempts at API endpoints.

CloudFront flat-rate pricing plans

CloudFront flat-rate pricing plans combine the Amazon CloudFront global content delivery network (CDN) with multiple AWS services and features into a monthly price with no overage charges, regardless of traffic spikes or attacks.

Flat-rate pricing plans include the following features for a simple monthly price:

- CloudFront CDN
- AWS WAF and DDoS protection
- Bot management and analytics
- Amazon Route 53 DNS
- Amazon CloudWatch Logs ingestion
- TLS certificate
- Serverless edge compute
- Amazon S3 storage credits each month

Plans are available in Free, Pro, Business, and Premium tiers to match your application's needs. Plans don't require an annual commitment to get the best available rates. Start with the Free plan and upgrade to access more capabilities and larger usage allowances.

Topics

- Benefits of CloudFront flat-rate pricing plans
- Features by pricing plan tier
- Monthly usage allowances
- Costs covered by your plan
- Reduce overall AWS costs with pricing plans
- Manage your flat-rate pricing plans
- Permissions
- Flat-rate pricing plan quotas
- Unsupported features

Benefits of CloudFront flat-rate pricing plans

The CloudFront pricing plan provides several key benefits:

- **Consolidated services and pricing**

  Combine multiple AWS services and features into a single plan for one flat rate. Designed to eliminate separate service purchases and upfront pricing calculations.
- **No overages**

  There are no overage charges regardless of traffic spikes or attacks.
- **Clear usage allowances**

  Each plan includes published usage allowances designed for optimal performance at that tier. Monitor your usage, receive proactive notifications, and upgrade based on your application's needs, with no long-term commitments.
- **Protect against DDoS attacks**

  CloudFront and AWS WAF absorb and block attacks before they reach your infrastructure. Reserves your compute, database, and infrastructure utilization only for legitimate traffic. Blocked DDoS attacks and requests blocked by AWS WAF never count against your usage allowance.
- **Reduce your overall AWS costs**

  Data transfer from AWS applications running on services such as Amazon S3, AWS Application Load Balancer (ALB), or Amazon API Gateway to CloudFront continues to be free. When you serve your AWS applications through CloudFront instead of directly to the internet, your flat-rate plan covers the data transfer costs between your applications and your viewers for a simple monthly price without the worry of overages. Fewer requests reaching your origin also reduces your costs on services that charge based on usage.

Features by pricing plan tier

Each pricing plan covers one CloudFront distribution with up to one apex (root) domain that combines essential features and services into one monthly price. Each plan also includes additional S3 storage credits.

Plans on higher tiers include all features from lower tier plans as well as additional features.

- **Free** – For hobbyists, learners, and developers getting started.
- **Pro** – Launch and grow small websites, blogs, and applications.
- **Business** – Protect and accelerate business applications.
- **Premium** – Scale and protect business and mission-critical applications.

Select a plan tier that includes features and configurations that you need for your applications. See the following features per pricing plan.

Pricing plan features

The following table shows the CloudFront, AWS WAF and DDoS, Amazon Route 53, Amazon CloudWatch, and Amazon S3 features included in each pricing plan tier.

Performance and Delivery	**Free**	**Pro**	**Business**	**Premium**
Global CDN Use CloudFront 750+ global edge locations as a massive, distributed, single point of entry for your web application. Accelerate static, dynamic, and non-cacheable applications.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Content caching Store copies of your content	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
in CloudFront 750+ edge loca tions worldwide , delivering it to users from the nearest l ocation. Reduces load times, protects your application from traffic spikes, and saves costs by serving repeated requests locally instead of from your application servers.				
Fast cache invalidations Remove or update cached content across all edge locations within seconds.	Yes	Yes	Yes	Yes

Performance and Delivery	Free	Pro	Business	Premium
Invalidations by cache tag Assign custom tags to your content and remove all cached co ntent sharing a specific tag, without tracking individual file paths.		Yes	Yes	Yes
Smart routing Intelligently routes users to the optimal edge location using real-time network data, and connects to your AWS origin th rough the AWS private network for better performance.	Yes	Yes	Yes	Yes

Performance and Delivery	Free	Pro	Business	Premium
Tiered caching Regional edge caches sit between edge locations and your application to store content longer, reducing load on your application and maintaining fast delivery.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Default caching rules Makes effective caching decisions to cache most web applications without custom configuration.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Custom caching rules Control how CloudFront caches content by specifying which request values to use, optimizing for your application's performance, personalization, and freshness needs using cache policies.			Yes	Yes
High-speed origin routing With Origin Shield, dynamic requests are routed from edge locations to your origin using CloudFront private network for high-performance path to your origin.				Yes

Performance and Delivery	Free	Pro	Business	Premium
Origin load reduction Adds an additional caching layer close to your web application using Origin Shield. Origin Shield consolidates requests from all edge locations, reducing load on your application particularly during traffic spikes.				Yes
Automatic origin failover Automatically routes traffic to a backup origin if your primary origin fails, maintaining high availability without disrupting users.				Yes

Performance and Delivery	Free	Pro	Business	Premium
Default origin request rules Control which information from viewer requests is automatically included in requests to your origin, using AWS managed origin request policies optimized for common scenarios.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Default response header rules Use AWS managed response headers policies to add or remove HTTP headers in responses to viewers, preconfigured for common security headers, CORS settings, and other standard use cases.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Custom origin request rules Create your own origin request policies to specify exactly which URL query strings, headers, and cookies are forwarded to your origin, enabling custom analytics and request handling.			✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Custom response header rules Create your own response header policies to control exactly which HTTP headers CloudFront adds or removes in responses to viewers, such as security headers, Content Security Policy (CSP), CORS settings, and custom application headers.			✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Number of cache behaviors Configure cache behaviors to control how CloudFront handles requests for specific URL patterns, including which origin serves the content, how content is cached, and whether HTTPS or signed URLs are required.	5	10	50	100
Security and Protection				
Always-on DDoS protection Protect against DDoS attacks that target your websites or applications.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Advanced DDoS Protection Identify and block DDoS attacks in seconds using the AntiDDoS AMR. AWS learns your unique application patterns to distinguish between attacks and natural surges from legitimate users.			✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Web Application Firewall (WAF) Protect against common application vulnerabilities and potential threats based on Amazon internal threat intelligence. Requests are blocked before reaching your servers.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Number of WAF rules Total number of security rules you can create and enable in your WAF configuration, including both custom rules and AWS Managed Rules.	5	25	50	75

Performance and Delivery	Free	Pro	Business	Premium
Protections for WordPress, PHP, and SQL data bases Use-case based security rules to protect common applications and operating systems like WordPress, PHP, SQL databases, Linux, and Windows.		✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
AI traffic analytics Monitor access patterns, request volumes, and popular paths of AI bots interacting with your content. AI bots are classified based on intent to detect activities like potentially unauthorized scraping used for training.		✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
IP-based rate limiting Automatically block IP addresses that exceed a configurable number of requests over a 5-minute period, protecting against HTTP flood attacks and Denial of Service (DoS) attempts.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Geographic traffic blocking Block requests from selected countries or regions.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
CAPTCHA challenge Require requests matching specific security rules to solve a CAPTCHA puzzle to prove that a human being is sending the request.		✓ Yes	✓ Yes	✓ Yes
Header-based threat filtering Create WAF security rules that filter threats based on HTTP request headers.		✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Regex-based threat filtering Create WAF security rules using regular expressions to match URI paths and HTTP request attributes.			✓ Yes	✓ Yes
JavaScript challenge Block automated threats by requiring browsers to complete JavaScript challenges that verify legitimate users.			✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Bot management and analytics (+AI bots)			✓ Yes	✓ Yes
Detect and analyze bot traffic with AWS WAF Bot Control for common bots. Provides controls to block, challenge, or allow unverified bots while identifying and distinguishing verified bots like search engines. Monitor AI bots and take action based on bot intent.				

Performance and Delivery	Free	Pro	Business	Premium
Custom WAF response Set a specific HTTP status code and optional custom HTML, plain text, or JSON response when requests are blocked by a rule.		✓ Yes	✓ Yes	✓ Yes
Header Insertion Add custom HTTP headers to requests that pass WAF inspection, enabling downstream applications to process requests differently or flag them for analysis.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Request body inspection Maximum size of HTTP request body content that can be inspected by AWS WAF for threats and malicious patterns.	16 KB	16 KB	64 KB	64 KB
Private origins within VPC Enhance security by keeping your application in a VPC private subnet, accessible only through your CloudFront distributions and hidden from the public internet, using VPC origins.			Yes	Yes

Performance and Delivery	Free	Pro	Business	Premium
Mutual TLS (origin) Restrict unauthorized access to your application (origin) using TLS-based client certificates, making sure only your authorized CloudFront distributions can establish connections to your application.			✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Origin Access Control (OAC) Maintain a private S3 bucket and only allow access through your designated CloudFront distribution, making sure your content is protected by your WAF rules, rate limits, and other security controls configured in your CloudFront distribution.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Free TLS certificate Free TLS certificate for your domain with automatic renewal through AWS Certificate Manager.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Signed URLs Create secure URLs that provide temporary access to private c ontent for specific users. Commonly used to share private documents with authorized users or grant secure access to protec ted content after payment verification.	Yes	Yes	Yes	Yes
Mutual TLS (mTLS) Restrict access to your applicati on using mTLS authentication, making sure only trusted clients with valid certificates can connect.				Yes

Performance and Delivery	Free	Pro	Business	Premium
Edge Compute				
Serverless edge compute Run lightweight JavaScript at the edge to modify URLs, HTTP headers, and request/response elements in milliseconds using CloudFront Functions. Lambda@Edge can also be used with all plan tiers, but unlike CloudFront Functions, Lambda@Edge invocations are billed on a pay-as-you-go basis.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Edge key-value store Store data at the edge using KeyValueStore for fast and dynamic content customization with CloudFront Functions.		Yes	Yes	Yes
CloudFront Functions network calls Make a network call to any origin on your distribution and act on the answer without leaving the edge.		Yes	Yes	Yes
Network and Protocol Support				

Performance and Delivery	Free	Pro	Business	Premium
IPv6 Deliver content over both modern IPv6 and traditional IPv4 connections from CloudFront to viewers and origins. Enables end-to-end IPv6 support for your applications.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
HTTP/2 Enable faster page loads through modern protocol features like multiplexing, header compression, and stream prioritization. Automatically used when supported by browsers and clients.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
HTTP/3 Deliver content using QUIC to browsers and clients that support it, enabling faster connections and improved performance. Particularly benefits mobile users and maintains connections when network conditions change.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
TLS 1.3 Deliver faster HTTPS connections through a handshake process that requires one round-trip compared to two in TLS 1.2. Reduces first byte latency by up to 33% compared to previous TLS versions. Enabled end-to-end for your applications.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
WebSockets Enable real-time, persistent two-way communication between browsers and servers. Ideal for AI chat applications, multi-player gaming, collaborative workspaces, and real-time data feeds like financial trading platforms.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Logging and Monitoring				

Performance and Delivery	Free	Pro	Business	Premium
Access Logs Access detailed CloudFront request logs to unders tand security and delivery traffic patterns, with Amazon CloudWatch Logs ingestion is included at no extra cost.		✓ Yes	✓ Yes	✓ Yes
WAF request logs Access detailed AWS WAF request logs to understan d security and delivery traffic patterns. Amazon CloudWatch Logs ingestion is included at no extra cost.		✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Security dashboard Monitor security events, investigate threats, and take immediate blocking actions using visual analytics without writing security rules. Pro and above includes visual log analyzer to quickly understand traffic patterns without querying logs.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
DNS				
Amazon Route 53 DNS Fast, reliable public authoritative DNS service using Route 53.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Records per Hosted Zone The maximum number of DNS records allowed in the hosted zone.	50	100	1000	5000
DNSSEC Protect your domain against DNS spoofing and man-in-the-middle attacks where attackers intercept DNS queries and redirect visitors to fake websites. Secures DNS traffic by cryptographically signing your DNS records.	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Storage				

Performance and Delivery	Free	Pro	Business	Premium
Amazon S3 storage Amazon S3 storage credits that offset any S3 Standard storage costs in your AWS account. Not limited to CloudFront content or subject to plan usage allowances.	5 GB	50 GB	1 TB	5 TB
Support and Reliability				

Performance and Delivery	Free	Pro	Business	Premium
24x7 account and billing support One-on-one responses to account and billing questions. If you have a paid support plan, you're eligible to receive support on all flat-rate plans.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Documentation and AWS Support forums Access product documentation, technical papers, best practices guides, AWS re:Post community forums, and service health information to help you plan and troubleshoot.	✓ Yes	✓ Yes	✓ Yes	✓ Yes

Performance and Delivery	Free	Pro	Business	Premium
Uptime SLA Service Level Agreements (SLA) for Amazon CloudFront, AWS WAF, Amazon Route 53, and Amazon CloudWatch provide service availability commitments. In the event AWS does not meet the associated SLA's commitment, you will be eligible to receive a service credit.			✓ Yes	✓ Yes

Monthly usage allowances

Each flat-rate plan includes a monthly usage allowance designed for optimal performance at that tier. **Usage allowances are not hard limits**—they represent the baseline usage your plan is designed to support. You can track your usage in the CloudFront console at any time and will receive automatic email notifications when you reach 50%, 80%, and 100% of your allowance.

Monthly usage allowances per plan tier

	Free	Pro	Business	Premium
Requests	1 M	10 M	125 M	500 M (configurable up to 6 B for a new flat-rate price)
Data transfer	100 GB	50 TB	50 TB	50 TB (configurable up to 600 TB for a new flat-rate price)

What counts toward your usage allowance

Blocked DDoS attacks and requests blocked by AWS WAF never count against your usage allowance. Only traffic that makes it past your AWS WAF security rules counts towards your allowance. This means attacks and unwanted traffic won't count against you, and you maintain the ability to define exactly what traffic your application blocks or allows.

What happens when usage exceeds the allowance

Plans are designed to be flexible and accommodate real-world traffic patterns like traffic variability, organic growth, and viral events. **Most importantly: you will not incur overage charges, regardless of how much you exceed your allowance.** If your usage exceeds your allowance through sustained growth over multiple months or through unusually high usage in a single month, your traffic delivery might be adjusted. **Most customers never experience performance adjustments.**

Here's how it works:

- **Your first traffic spike up to 3x your monthly allowance won't affect your service that month.** This one-time accommodation handles unexpected events like viral content or successful product launches without penalizing your success.
- **Sustained usage above your allowance is evaluated over 2-3 months or more, not immediately.** Minor fluctuations and moderate growth are expected and accommodated. Only substantial, sustained excess usage indicates your application has outgrown your tier.

- **You'll receive notifications each month as you approach and exceed your allowance.** If your usage consistently and significantly exceeds your plan, we recommend upgrading to a tier that matches your growth and makes sure optimal performance as you scale. You control your performance by upgrading when your baseline usage patterns change.
- **If you continue to substantially exceed your plan's usage allowance without upgrading, your traffic delivery might be adjusted.** For example, your traffic might be served from fewer or more distant edge locations, or performance might be adjusted. The degree of adjustment is proportional—small excess usage sees minimal changes, larger sustained excess sees more noticeable changes. Upgrading restores full performance.

Most customers never experience performance adjustments. Plans are designed to accommodate normal growth patterns.

Configurable usage allowances on the Premium plan

The Premium plan offers configurable monthly usage allowances per distribution. When you subscribe to or manage a Premium plan, you can select a higher monthly usage allowance from the following levels:

Premium usage levels

Premium usage level	Monthly data transfer	Monthly requests	Flat-rate price per month
Premium (default)	50 TB	500 M	$1,000
Premium	75 TB	750 M	$1,450
Premium	125 TB	1.25 B	$2,250
Premium	200 TB	2 B	$3,500
Premium	350 TB	3.5 B	$6,000
Premium	600 TB	6 B	$10,000

When you select a higher usage level, your monthly price increases and your monthly usage allowance increases accordingly. The features and services included in the Premium plan remain

the same at every usage level. You are only changing your usage allowance and flat-rate price. The same usage allowance policies apply at all usage levels. For details, see What happens when usage exceeds the allowance.

You can change your usage level at any time in the CloudFront console. When you increase your usage level, changes take effect immediately. Your price and usage allowance are prorated. When you decrease your usage level, the change takes effect at the beginning of the next billing cycle.

If your application's baseline usage exceeds 6 B requests or 600 TB per month, contact us for custom pricing.

> **Note**
>
> Configurable usage allowances are available only on the Premium plan. Usage allowances on Free, Pro, and Business plans are not configurable.

Eligibility based on historical usage

Your historical CloudFront usage might affect your eligibility to sign up for or downgrade to specific plan tiers. If your recent usage exceeds a plan tier's usage allowances, you might need to select a higher tier that better aligns with your workload.

Costs covered by your plan

Your plan covers costs for:

- Your CloudFront distribution
- The AWS WAF web ACL, custom rules, AWS Managed Rules, and request fees for the web ACL associated with your distribution
- CloudWatch Logs ingestion for your distribution's CloudFront access logs and associated WAF logs
- The Route 53 hosted zone, DNS records, and DNS queries when attached to your distribution's plan

You will also receive S3 credits to offset S3 Standard storage usage in your payer account, whether or not an S3 bucket is used as an origin for your CloudFront distribution.

Route 53 DNS management and your plan

If you use Route 53 for DNS and attach the zone to your plan, your flat-rate plan can include your Route 53 hosted zone costs. You can attach the zone to your plan in the **Manage Plan** section of your CloudFront distribution. When your zone is attached to the plan, your plan covers your hosted zone's standard costs, including the monthly hosted zone fee, DNS records, and DNS query fees subject to respective allowances per tier, provided below. The hosted zone must meet the following requirements:

- Exist in the same AWS account as your CloudFront distribution
- Maintain the number of records allowed per hosted zone for your plan tier
- Cover the domain used by your CloudFront distribution

If your hosted zone is not attached to your plan, it will remain on pay-as-you-go pricing, where you're responsible for all standard Route 53 costs.

Understanding monthly DNS query allowances

When your hosted zone is attached to your plan, you get:

1. DNS queries to ALIAS records pointing to your CloudFront distribution and other supported AWS services
2. An additional monthly allowance for other DNS record types

	Free	Pro	Business	Premium
DNS queries to ALIAS records (CloudFront and other supported AWS services) per month	No limit	No limit	No limit	No limit
	1 M	5 M	20 M	100 M

	Free	Pro	Business	Premium
Additional DNS query allowance per month				

> **Note**
>
> To maximize your plan benefits, use ALIAS records to point to your CloudFront distribution. ALIAS records pointing to CloudFront and other supported AWS services don't count against your monthly DNS query allowance. All other DNS queries, including CNAME records to CloudFront, count against your DNS query allowance.

Exceeding DNS query allowances

If your DNS query usage exceeds your plan's monthly allowance, AWS might notify you. At that point, you can detach your hosted zone from the plan in the **Manage Plan** section of your CloudFront distribution to return the hosted zone to pay-as-you-go pricing. If you do not detach your hosted zone after receiving this notification, AWS might automatically transition the hosted zone to pay-as-you-go pricing. When a hosted zone moves to pay-as-you-go pricing, you are responsible for all standard Route 53 costs. Your CloudFront distribution and all other plan benefits continue unchanged.

Reduce overall AWS costs with pricing plans

CloudFront flat-rate pricing plans can reduce your overall AWS costs in three ways:

First, data transfer costs between CloudFront and your AWS applications running on services such as Amazon S3, AWS Application Load Balancer (ALB), or Amazon API Gateway are automatically waived. When you serve your AWS applications through CloudFront instead of directly to the internet, your flat-rate plan covers the data transfer costs between your applications and your viewers for a simple monthly price without the worry of overages.

Second, CloudFront reduces your compute and database costs by protecting your application infrastructure and reducing the number of requests reaching your origin. It serves cached content from edge locations or regional edge caches, collapses duplicate requests, and blocks malicious

and unwanted traffic before it reaches your backend services. This means fewer requests hitting your application servers, databases, and other AWS services that charge based on usage, which reduces your costs.

Finally, each plan includes Amazon S3 Standard storage credits to offset storage usage for your AWS account.

To maximize these savings, configure your AWS origins to only accept traffic from CloudFront. For S3, use Origin Access Control OAC with private buckets to grant access to your designated CloudFront distribution. For Application Load Balancer, Network Load Balancer, and Amazon EC2 instances in private subnets, restrict access to your designated CloudFront distribution using VPC Origins.

Manage your flat-rate pricing plans

You can manage your CloudFront flat-rate pricing plans using the CloudFront console, or programmatically using the AWS Command Line Interface (AWS CLI) or the PricingPlanManager API. This section describes how to subscribe distributions to pricing plans, change plan tiers, and cancel subscriptions using the CloudFront console.

For information about managing pricing plans programmatically, see Getting started with the PricingPlanManager API.

Subscribe a new distribution to a pricing plan

When you create a new distribution, you can subscribe to a pricing plan.

To subscribe a new distribution to a pricing plan

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, then follow the steps to create a distribution.
3. Choose your distribution's pricing plan. Note that some features are not available per pricing plan tier. Review the features per plan and choose the pricing plan that you need for your application.
4. Complete the steps to create your distribution.

Subscribe an existing distribution to a pricing plan

When you update a distribution, you can subscribe to a pricing plan. Before choosing a pricing plan, make sure that your distribution configuration is compatible with the plan that you want.

> **Tip**
>
> If your current distribution uses any unsupported features, you must disable those features before you can subscribe to the pricing plan. This includes disabling features like real-time access logs.

After your distribution configuration is compatible, you can choose your desired pricing plan while updating a distribution.

To subscribe an existing distribution to a pricing plan

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, then follow the steps to update an existing distribution.
3. Choose your distribution's pricing plan. Note that some features are not available per pricing plan tier. Review the features per plan and choose the pricing plan that you need for your application.
4. Complete the steps to update your distribution.

Upgrade a pricing plan

We recommend that you upgrade a plan if you're approaching or have exceeded your monthly usage allowance, or if you want to enable a feature that is available in the next tier.

When you upgrade to a higher plan tier, changes take effect immediately. Your price and usage allowance are prorated. Your distribution and associated resources will have access to the available features and higher usage allowance of your new plan.

To upgrade a pricing plan

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose your distribution that is subscribed to an existing pricing plan.
4. Follow the prompts to upgrade your distribution's pricing plan.
5. Complete the steps to update an existing distribution.

Change your Premium plan usage level

If you are on the Premium plan, you can change your usage level to increase or decrease your usage allowance.

When you increase your usage level, the change takes effect immediately and your price is prorated. When you decrease your usage level, the change takes effect at the beginning of the next billing cycle.

To change your Premium plan usage level

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose your distribution that is subscribed to a Premium pricing plan.
4. Choose the **Manage Plan** button.
5. In the **Manage plan** section, choose **Change plan**.
6. Follow the prompts to change your usage level for the Premium pricing plan. The console displays the available usage levels and corresponding monthly prices.

Downgrade a pricing plan

We recommend that you downgrade to a lower plan tier if you don't need the additional features on your existing tier. For example, you might downgrade if you expect your application will experience lower traffic.

If you downgrade to a lower tier, your billing changes will take effect at the beginning of the next billing cycle.

If your distribution currently exceeds the usage allowance for a plan, you can downgrade after your usage is within the usage allowance for your desired tier. To avoid being charged for your existing plan tier at the next billing cycle, downgrade before the end of the month.

To downgrade a pricing plan

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose your distribution that is subscribed to an existing pricing plan.
4. Follow the prompts to downgrade your distribution's pricing plan. If you have unsupported features, you must either remove the feature or resource from the distribution.
5. Complete the steps to update an existing distribution.

Cancel a pricing plan

When you cancel a paid pricing plan, you will maintain your flat-rate price through the end of your current billing cycle. Your distribution and all associated plan resources will then switch to pay-as-you-go pricing at the start of the next billing cycle. Free plans are canceled immediately.

To cancel a pricing plan

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose your distribution that is subscribed to an existing pricing plan.
4. Follow the prompts to cancel your distribution's pricing plan.
5. Complete the steps to update an existing distribution.

Cancel a pending plan change

If you downgraded or canceled your flat-rate pricing plan, you must wait until the end of the current billing cycle before your changes are in effect. To keep your existing flat-rate pricing plan, upgrade, or downgrade your pricing plan again, you must first cancel your pending plan change.

To cancel a pending pricing plan change

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose your distribution that is subscribed to an existing pricing plan.
4. Follow the prompts to cancel your distribution's pending plan change.
5. Choose the pricing plan that you want for your distribution.
6. Complete the steps to update an existing distribution.

Deleting a distribution with a pricing plan

You can't delete a distribution that is subscribed to a pricing plan. You must first cancel the pricing plan. Paid pricing plan cancellations take effect after the current billing cycle. Free plans are canceled immediately. After the plan is canceled, you can delete the distribution.

To delete a distribution with a pricing plan

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Follow the previous steps to cancel the distribution's pricing plan.
4. Follow the steps to delete the distribution.

> **Note**
>
> You can disable a distribution that is subscribed to a pricing plan, but you will still incur charges for that plan. To stop incurring charges for your plan, you must first cancel it.

Permissions

To view or manage pricing plan subscriptions for your CloudFront distributions, you must have the required permissions. For more information, see AWS managed policy: CloudFrontFullAccess and AWS managed policy: CloudFrontReadOnlyAccess.

Flat-rate pricing plan quotas

The following table shows the quotas and restrictions for CloudFront flat-rate pricing plans.

> **Note**
>
> These quotas can't be increased for your AWS account.

Account-level quotas	Quotas
Pricing plans per AWS account	100
Free plans per AWS account	3
Apex-level domains per plan	1

Unsupported features

Before you can associate a distribution with a pricing plan, you must make sure that certain features are disabled and associations are removed.

> **Notes**
>
> - If your distribution or account has any of these restrictions, you must resolve them before you can use pricing plans. After you make changes to your distribution, wait for the changes to propagate to all edge locations.
> - You must have an AWS WAF Web ACL associated with your distribution if you're using a pricing plan. This resource cannot be removed or disassociated from your distribution unless you switch to pay-as-you-go pricing for that distribution.

Unsupported features

You can't subscribe distributions to a pricing plan if their configuration contains the following unsupported features. You can disable the unsupported feature and use an alternative option, or keep pay-as-you-go for your distribution.

Unsupported features	Alternative options	AWS service
Multi-tenant distributions	Use a standard distribution or pay-as-you-go pricing	CloudFront
Continuous deployment and Staging distributions	Use pay-as-you-go pricing	CloudFront
Anycast IP list configuration	Use pay-as-you-go pricing	CloudFront
Real-time access logs	Use standard access logs or pay-as-you-go pricing	CloudFront
Targeted Bots	Use common bots or pay-as-you-go pricing	AWS WAF
Partner Managed Rules	Use pay-as-you-go pricing	AWS WAF
Account Creation Fraud Prevention	Use pay-as-you-go pricing	AWS WAF
Account Takeover Protection	Use pay-as-you-go pricing	AWS WAF
Rule Groups	Create individual rules (rule groups are shared AWS WAF rules that can be applied to a web ACL, similar to policies on CloudFront)	AWS WAF

Unsupported features	Alternative options	AWS service
Legacy features		
ForwardedValues configuration	Use Origin request policies	CloudFront
Dedicated IP/SSL	Use pay-as-you-go pricing	CloudFront
Field level encryption	Use pay-as-you-go pricing	CloudFront
AWS Identity and Access Management (IAM) server certificates	Use AWS Certificate Manager (ACM) certificates	CloudFront
Origin access identity (OAI)	Use Origin access control (OAC)	CloudFront
Legacy cache settings	Use cache policies and origin request policies.	CloudFront

Unsupported associations

You can't subscribe a distribution to a pricing plan if the distribution is already associated with any of the following resources that are *already associated* with other distributions. Resources that are associated to a distribution that is subscribed to a pricing plan can only be used for that distribution. For example, if you have a CloudFront function that is using a key value store, neither the function nor the key value store can be shared for a distribution that is on a pricing plan.

- CloudFront Functions
- CloudFront Functions associated with a key value store
- AWS WAF Web ACLs

To subscribe a distribution to a pricing plan, either remove the associated resource or replace it with another one.

Account-level constraints

AWS accounts are not eligible for pricing plans if they meet any of the following conditions:

- You reached the maximum number of subscriptions allowed. See Flat-rate pricing plan quotas.
- Your account is using AWS Free Tier.

Resource-level constraints

Distributions are not eligible for pricing plans if they meet any of the following conditions:

- Your distribution has enabled AWS Shield Advanced
- Your distribution has enabled the Firewall Manager Service for your web ACL. Firewall Manager won't manage your CloudFront distribution's WebACL in a pricing plan.

Additional features that can affect your pricing plan

Flat-rate pricing plans enable you to pay a flat-rate for your CloudFront distribution and the features listed above that are both included in your plan and associated with your CloudFront distribution. All other features might incur additional charges, including but not limited to the following:

CloudFront

- Lambda@Edge function invocations

AWS WAF

- CAPTCHA puzzles created using the JavaScript API are billed using pay-as-you-go pricing. CAPTCHA responses configured in your AWS WAF rules (the most common use case) are included in your plan at no additional charge.

Route 53

- Route 53 DNSSEC has an AWS KMS cost
- Route 53 IP (CIDR) blocks (the first 1,000 are free per AWS account)
- Route 53 Health Checks (the first 50 are free per AWS account)

Logging features

- Route 53 DNS Query Logs, CloudFront Functions logs, and CloudFront Connection Function Logs
- AWS WAF log delivery to Amazon S3
- CloudFront or AWS WAF log delivery to Amazon Data Firehose
- Additional CloudWatch metrics for CloudFront
- CloudFront access logs in Parquet format

> **Note**
>
> Your plan includes Amazon CloudWatch Logs ingestion for CloudFront standard logs (access logs) and WAF logs for no added costs. All other CloudWatch costs such as storage and querying are not covered by your plan. All other logs are also billed separately.

> **Note**
>
> Your plan includes public authoritative DNS from Route 53. When your Route 53 hosted zone is attached to your pricing plan, your plan covers your hosted zone's standard costs, including the monthly hosted zone fee, DNS records, and DNS query fees subject to respective allowances per tier. All other costs from Route 53 usage and features not listed above as included in your plan are not covered by your plan.

Pricing plans versus pay-as-you-go pricing

Flat-rate plans and pay-as-you-go pricing offer different advantages based on your needs. With flat-rate plans, you pay one price that includes multiple AWS services like CloudFront, AWS WAF, Route 53, and CloudWatch Logs ingestion and never face overage charges, even during traffic spikes or attacks.

With pay-as-you-go pricing, you're billed separately for each service and feature based on your actual usage. While this provides complete flexibility in service selection and configuration, your costs can vary month to month based on traffic patterns, and you will need to monitor usage across multiple services to manage costs.

Flat-rate plans are ideal if you want combined monthly billing, simplified service configuration, and built-in security features without worrying about overage charges. Pay-as-you-go pricing is a better choice if you need complete control over individual service features, custom configurations, access to features not available in flat-rate plans, or if you expect to handle large, predictable traffic spikes. Amazon CloudFront flat-rate pricing plans cannot be combined with any other offers, promotions, or discounts.

Configure distributions

You create an Amazon CloudFront distribution to tell CloudFront from where you want content to be delivered, and the details about how to track and manage content delivery.

When you create your CloudFront distribution, CloudFront automatically configures most distribution settings for you, based on your content origin type. For more details about the preconfigured settings, see Preconfigured distribution settings reference. Optionally, you can choose to manually edit your distribution settings. For more information, see All distribution settings reference.

The following settings can be configured:

- **Your content origin**—The Amazon S3 bucket, AWS Elemental MediaPackage channel, AWS Elemental MediaStore container, Elastic Load Balancing load balancer, or HTTP server from which CloudFront gets the files to distribute. You can specify any combination of up to 25 origins for a single distribution.
- **Access**—Whether you want access to the files to be available to everyone or restricted to some users.
- **Security**—Whether you want to enable AWS WAF protection and require users to use HTTPS to access your content. For multi-tenant distributions, only AWS WAF V2 web access control lists (ACLs) are supported.
- **Cache key**—Which values, if any, you want to include in the *cache key*. The cache key uniquely identifies each file in the cache for a given distribution.
- **Origin request settings**—Whether you want CloudFront to include HTTP headers, cookies, or query strings in requests that it sends to your origin.
- **Geographic restrictions**—Whether you want CloudFront to prevent users in selected countries from accessing your content.
- **Logs**—Whether you want CloudFront to create standard logs or real-time access logs that show viewer activity.

For more information, see All distribution settings reference.

For the current maximum number of distributions that you can create for each AWS account, see General quotas on distributions. There is no maximum number of files that you can serve per distribution.

You can use distributions to serve the following content over HTTP or HTTPS:

- Static and dynamic download content, such as HTML, CSS, JavaScript, and image files, using HTTP or HTTPS.
- Video on demand in different formats, such as Apple HTTP Live Streaming (HLS) and Microsoft Smooth Streaming. (For multi-tenant distributions, Smooth Streaming is not supported.) For more information, see Deliver video on demand with CloudFront.
- A live event, such as a meeting, conference, or concert, in real time. For live streaming, you can create the distribution automatically by using a CloudFormation stack. For more information, see Deliver video streaming with CloudFront and AWS Media Services.

The following topics provide more details about CloudFront distributions and how to configure them to meet your business needs. For more information about creating a distribution, see Create a distribution.

Topics

- Understand how multi-tenant distributions work
- Create a distribution
- Preconfigured distribution settings reference
- All distribution settings reference
- Test a distribution
- Update a distribution
- Tag a CloudFront resource
- Delete a distribution
- Use various origins with CloudFront distributions
- Enable IPv6 for CloudFront distributions
- Use CloudFront continuous deployment to safely test CDN configuration changes
- Use custom URLs by adding alternate domain names (CNAMEs)
- Use WebSockets with CloudFront distributions
- Request Anycast static IPs to use for allowlisting
- Using gRPC with CloudFront distributions

Understand how multi-tenant distributions work

You can create CloudFront multi-tenant distributions with settings that can be reused across multiple distribution tenants. With a multi-tenant distribution, you can have CloudFront configure your distribution settings for you, based on your content origin type. For more details about the preconfigured settings, see Preconfigured distribution settings reference.

To move an existing standard distribution to a multi-tenant distribution, see Migrate to a multi-tenant distribution.

Benefits of using a multi-tenant distribution instead of a standard distribution include:

- Reducing operational burden.
- Reusable configurations for web admins and software providers to manage CloudFront distribution for multiple web applications that deliver content to end users.
- Enhanced integrations with other AWS services to deliver automated certificate management, unified security controls, and hassle-free configuration control at scale.
- Maintaining consistent resource patterns across your implementations. Define settings that must be shared and then specify customizations for settings to override.
- Customizable origin and security settings to meet specific needs at the distribution tenant level.
- Organize your distribution tenants into different tiers. For example, if some distribution tenants require Origin Shield and some do not, you can group distribution tenants into different multi-tenant distributions.
- Sharing a common DNS configuration across multiple domains.

Unlike a standard distribution, a multi-tenant distribution cannot be accessed directly because it has no CloudFront routing endpoint. Therefore, it must be used in conjunction with a connection group and one or more distribution tenants. While standard distributions have their own CloudFront endpoint and can be directly accessed by end users, they cannot be used as a template for other distributions.

For information about multi-tenant distribution quotas, see Quotas on multi-tenant distributions.

Topics

- How it works
- Terms

- Unsupported features
- Distribution tenant customizations
- Request certificates for your CloudFront distribution tenant
- Serve distribution tenant domains without a certificate
- Create custom connection group (optional)
- Serve domains on specific IP addresses with address-based selection
- Serve viewers that don't send SNI with a viewer fallback
- Migrate to a multi-tenant distribution

How it works

In a *standard distribution*, the distribution contains all the settings that you want to enable for your website or application, such as the origin configurations, cache behaviors, and security settings. If you wanted to create a separate website and use many of the same settings, you would need to create a new distribution each time.

CloudFront multi-tenant distributions are different in that you can create an initial multi-tenant distribution. For each new website, you create a distribution tenant that automatically inherits the defined values of its source distribution. Then, you customize specific settings for your distribution tenant.

Overview

1. To get started, you first create a multi-tenant distribution. CloudFront configures your distribution settings for you, based on your content origin type. You can customize settings for all origins except VPC origins. VPC origins settings are customized on the VPC origin resource itself. For more information about the multi-tenant distribution settings that you can customize, see Preconfigured distribution settings reference.
   - The TLS certificate that you use for the multi-tenant distribution can be inherited by your distribution tenants. The multi-tenant distribution itself is not routable, so it will not have a domain name associated with it.
2. By default, CloudFront creates a connection group for you. The connection group controls how viewer requests for content connect to CloudFront. You can customize some routing settings in the connection group.

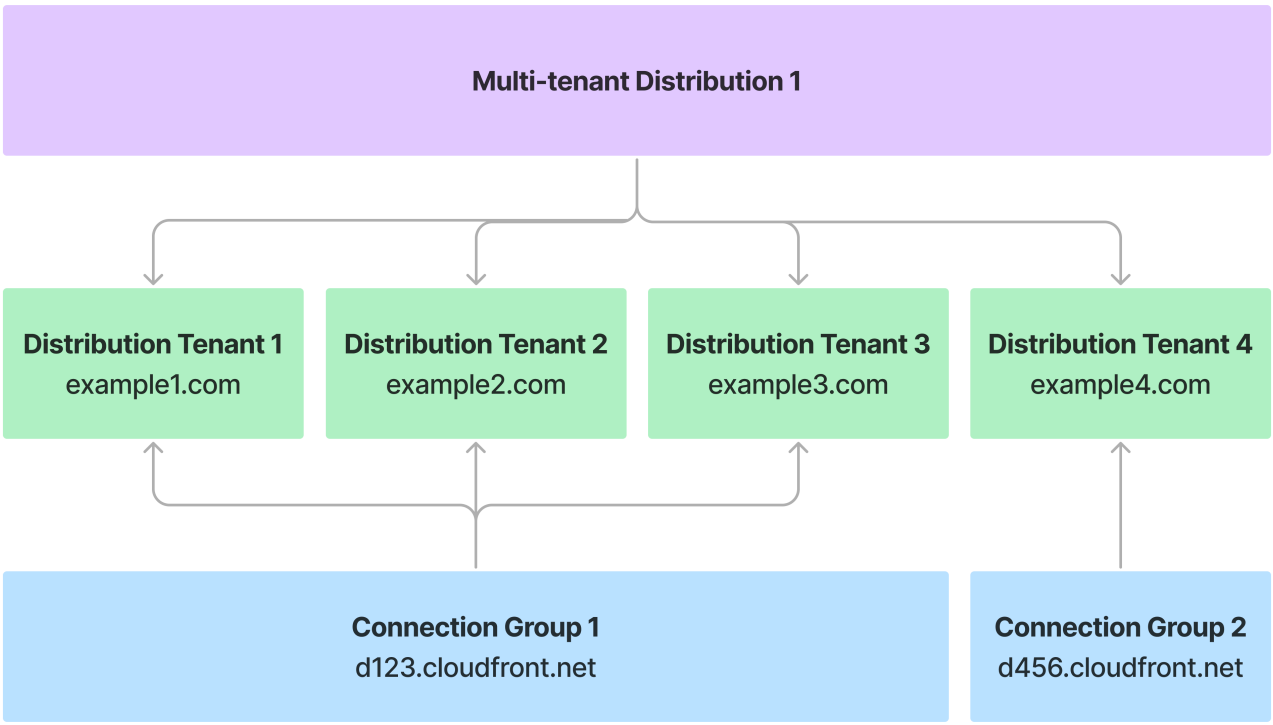
You can change this by manually creating your own connection group. For more information, see Create custom connection group (optional).

3. Then, you create one or more distribution tenants. The distribution tenant is the "front door" for viewers to access your content. Each distribution tenant references the multi-tenant distribution and is automatically associated with the connection group that CloudFront created for you. The distribution tenant supports an individual domain or subdomain.
4. You can then customize some distribution tenant settings, such as vanity domains and origin paths. For more information, see Distribution tenant customizations.
5. Finally, you must update the DNS record in your DNS host to route traffic to the distribution tenant. To do this, get the CloudFront endpoint value from your connection group and create a CNAME record that points to the CloudFront endpoint.

Example

The following graphic demonstrates how a multi-tenant distribution, distribution tenants, and connection groups work together to deliver content for your viewers for multiple domains.

1. The multi-tenant distribution defines the inherited settings for each distribution tenant. You use the multi-tenant distribution as a template.
2. Each distribution tenant created from the multi-tenant distribution has its own domain.
3. The distribution tenants are automatically added to the connection group that CloudFront created for you when you created the multi-tenant distribution. Connection groups control how viewer requests are connected to the CloudFront network.



For detailed multi-tenant distribution creation instructions, see Create a CloudFront distribution in the console.

Terms

The following concepts describe components of multi-tenant distributions:

Multi-tenant distribution

A blueprint, multi-tenant distribution that specifies all shared configuration settings for any distribution tenants, including cache behavior, security protections, and origins. Multi-tenant distributions cannot serve traffic directly. They must be used in conjunction with connection groups and distribution tenants.

Standard distribution

A distribution that does not have multi-tenant functionality. These distributions are best for supporting single websites or apps.

Distribution tenant

A distribution tenant inherits the multi-tenant distribution configuration. Some configuration settings can be customized at the distribution tenant level. To serve HTTPS, the distribution tenant must have a valid TLS certificate, which can be inherited from the multi-tenant distribution as long as it covers the distribution tenant domain or subdomain. A distribution tenant can also serve a domain over HTTP without a certificate. For more information, see Serve distribution tenant domains without a certificate.

The distribution tenant must be associated with a connection group. CloudFront creates a connection group for you when you create a distribution tenant, and automatically assigns any distribution tenants to that connection group.

Multi-tenancy

You can use the multi-tenant distribution to serve content across multiple domains, while sharing configuration and infrastructure. This approach allows different domains (called tenants) to share common settings from the multi-tenant distribution, while maintaining their own customizations.

Connection group

Provides the CloudFront routing endpoint that serves content to viewers. You must associate each distribution tenant to a connection group to get the corresponding CloudFront routing endpoint for the CNAME record that you create for your distribution tenant domain or subdomain. Connection groups can be shared across multiple distribution tenants. Connection groups manage routing settings for distribution tenants, such as IPv6 and Anycast IP list settings.

Parameters

A list of key-value pairs for placeholder values, such as origin paths and domain names. You can define parameters in your multi-tenant distribution and provide values for those parameters at the distribution tenant level. You choose whether the parameter values are required to be entered for the distribution tenant.

If you do not provide a value for an optional parameter in a distribution tenant, then the default value from the multi-tenant distribution is used as the value.

CloudFront routing endpoint

Canonical DNS for the connection group, such as `d123.cloudfront.net`. Used in the CNAME record for your distribution tenant domain or subdomain.

Customizations

You can customize your distribution tenants so that they use *different* settings from the multi-tenant distribution. For each distribution tenant, you can specify a different AWS WAF web access control list (ACL), TLS certificates, and geographic restrictions.

Unsupported features

The following features can't be used with a multi-tenant distribution. If you want to create a new multi-tenant distribution using the same settings as your standard distribution, note that some settings aren't available.

Notes

- Currently, AWS Firewall Manager policies only apply to your standard distributions. Firewall Manager will add support for multi-tenant distributions in a future release.
- Unlike standard distributions, you specify your domain name (alias) at the *distribution tenant* level. For more information, see Request certificates for your CloudFront distribution tenant and the CreateDistributionTenant API operation.

- Continuous deployment
- Origin access identity (OAI) – Use origin access control (OAC) instead.
- Dedicated IP custom SSL support – Only the `sni-only` method is supported.
- AWS WAF Classic (V1) web ACL – Only AWS WAF V2 web ACLs are supported.
- Standard logging (legacy)
- Minimum TTL
- Default TTL
- Maximum TTL
- ForwardedValues
- PriceClass
- Trusted Signers
- Smooth streaming

- AWS Identity and Access Management (IAM) server certificates
- Dedicated IP addresses
- Minimum Protocol Version SSLv3
- Network calls from CloudFront Functions

The following settings can't be configured in a multi-tenant distribution or distribution tenant. Instead, set the values that you want in a connection group. All distribution tenants associated in the connection group will use these settings. For more information, see Create custom connection group (optional).

- Enable IPv6 (viewer requests)
- Anycast static IP list

Distribution tenant customizations

When using a multi-tenant distribution, your distribution tenants inherit the multi-tenant distribution configuration. However, you can customize some settings at the distribution tenant level.

You can customize the following:

- **Parameters** – Parameters are key-value pairs that you can use for the origin domain or origin paths. See How parameters work with distribution tenants.
- **AWS WAF web ACL (V2)** – You can specify a separate web ACL for the distribution tenant, which will *override* the web ACL used for the multi-tenant distribution. You can also disable this setting for a specific distribution tenant, which means the distribution tenant won't inherit the web ACL protections from the multi-tenant distribution. For more information, see AWS WAF web ACL.
- **Geographic restrictions** – Geographic restrictions that you specify for a distribution tenant will *override* any geographic restrictions for the multi-tenant distribution. For example, if you block Germany (DE) in your multi-tenant distribution, all associated distribution tenants will also block DE. However, if you allow DE for a specific distribution tenant, that distribution tenant settings will override the settings for the multi-tenant distribution. For more information, see Restrict the geographic distribution of your content.
- **Invalidation paths** – Specify the file paths to the content that you want to invalidate for the distribution tenant. For more information, see Invalidate files.

- **Custom TLS certificates** – AWS Certificate Manager (ACM) certificates that you specify for distribution tenants are supplemental to the certificate provided in the multi-tenant distribution. However, if the same domain is covered by both the multi-tenant distribution and distribution tenant certificates, then the tenant certificate is used. The distribution tenant uses the security policy of the multi-tenant distribution with its certificate, even when the multi-tenant distribution uses the default CloudFront certificate. For more information, see Security policy (minimum SSL/TLS version) and Request certificates for your CloudFront distribution tenant.
- **Domain names** – You must specify at least one domain name per distribution tenant.

How parameters work with distribution tenants

A parameter is a key-value pair that you can use for placeholder values. Define the parameters that you want to use in your multi-tenant distribution and specify whether they're required.

When you define parameters in your multi-tenant distribution, you choose whether those parameters are required to be entered at the distribution tenant level.

- If you define the parameters as *required* in the multi-tenant distribution, then they must be entered at the distribution tenant level. (They are not inherited).
- If the parameters are *not required*, then you can provide a default value in the multi-tenant distribution that is inherited by the distribution tenant.

You can use parameters in the following properties:

- Origin domain name
- Origin path

In the multi-tenant distribution, you can define up to two parameters for each of the preceding properties.

Example parameters

See the following examples for using parameters for the domain name and origin path.

Domain name parameters

In the multi-tenant distribution configuration, you can define a parameter for the origin domain name like the following examples:

Amazon S3

- {{*parameter1*}}.amzn-s3-demo-logging-bucket.s3.eusc-de-east-1.amazonaws.com
- {{*parameter1*}}-amzn-s3-demo-logging-bucket.s3.eusc-de-east-1.amazonaws.com

Custom origins

- {{*parameter1*}}.lambda-url.eusc-de-east-1.on.aws
- {{*parameter1*}}.mediapackagev2.ap-south-1.amazonaws.com

When you create a distribution tenant, specify the value to use for *parameter1*.

```
"Parameters": [
  {
    "Name": "parameter1",
    "Value": "mycompany-website"
  }
]
```

Using the previous examples specified in the multi-tenant distribution, the origin domain name for the distribution tenant resolves to the following:

- *mycompany-website*.amzn-s3-demo-bucket3.s3.eusc-de-east-1.amazonaws.com
- *mycompany-website*-amzn-s3-demo-bucket3.s3.eusc-de-east-1.amazonaws.com
- *mycompany-website*.lambda-url.eusc-de-east-1.on.aws
- *mycompany-website*.mediapackagev2.ap-south-1.amazonaws.com

Origin path parameters

Similarly, you can define parameters for the origin path in the multi-tenant distribution like the following examples:

- /{{*parameter2*}}
- /{{*parameter2*}}/test

- /public/{{*parameter2*}}/test
- /search?name={{*parameter2*}}

When you create a distribution tenant, specify the value to use for *parameter2*.

```
"Parameters": [
  {
    "Name": "parameter2",
    "Value": "myBrand"
  }
]
```

Using the previous examples specified in the multi-tenant distribution, the origin path for the distribution tenant resolves to the following:

- /*myBrand*
- /*myBrand*/test
- /public/*myBrand*/test
- /search?name=*myBrand*

Example

You want to create multiple websites (tenants) for your customers, and you need to make sure that each distribution tenant resource uses the correct values.

1. You create a multi-tenant distribution and include two parameters for the distribution tenant configuration.
2. For the origin domain name, you create a parameter named *customer-name* and specify that it's required. You enter the parameter before the S3 bucket, so that it appears as:

   {{*customer-name*}}.amzn-s3-demo-bucket3.s3.eusc-de-east-1.amazonaws.com.
3. For origin path, you create a second parameter named *my-theme*, and specify that it's optional, with a default value of *basic*. Your origin path appears as: /{{*my-theme*}}
4. When you create a distribution tenant:
   - For domain name, you must specify a value for *customer-name*, because it's marked as required in the multi-tenant distribution.

- For origin path, you can optionally specify a value for *my-theme* or use the default value.

Request certificates for your CloudFront distribution tenant

When you create a distribution tenant, the tenant inherits the shared AWS Certificate Manager (ACM) certificate from the multi-tenant distribution. This shared certificate provides HTTPS for all tenants associated with the multi-tenant distribution.

When you create or update a CloudFront distribution tenant to add domains, you can add a managed CloudFront certificate from ACM. CloudFront then gets an HTTP-validated certificate from ACM on your behalf. You can use this tenant-level ACM certificate for custom domain configurations. CloudFront streamlines the renewal workflow to help keep certificates up-to-date and secure content delivery uninterrupted.

> **Note**
>
> You own the certificate, but it can *only* be used with CloudFront resources and the private key *cannot* be exported.

You can request the certificate when you create or update the distribution tenant.

Choose how to serve certificate validation tokens

When you request a managed certificate for a distribution tenant through the API or AWS CLI, specify `ValidationTokenHost` in `ManagedCertificateRequest`. Choose the value based on who serves the HTTP validation token.

Both options request a CloudFront-managed ACM certificate. The parameter only controls where the validation token is served.

CloudFront-hosted (`cloudfront`)

CloudFront automatically serves the HTTP validation token. Choose this option when the domain has no existing traffic, or when you can point its DNS to CloudFront immediately.

Self-hosted (`self-hosted`)

You serve the HTTP validation token from your existing infrastructure. Choose this option when you're migrating a domain with existing traffic. The certificate can be issued before you point

DNS to CloudFront. You can upload the token file to your existing server, or configure an HTTP 301 (permanent) redirect that forwards validation requests to the ACM validation endpoint.

For instructions on completing domain ownership setup, see Complete domain setup. If you chose `cloudfront`, select **I don't have traffic**. If you chose `self-hosted`, select **I have existing traffic**.

For more information about the API values, see ManagedCertificateRequest in the *CloudFront API Reference*.

Topics

- Add a domain and certificate (distribution tenant)
- Complete domain setup
- Point domains to CloudFront
- Domain considerations (distribution tenant)
- Wildcard domains (distribution tenant)

Add a domain and certificate (distribution tenant)

The following procedure shows you how to add a domain and update the certificate for a distribution tenant.

To add a domain and certificate (distribution tenant)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Under **SaaS**, choose **Distribution tenants**.
3. Search for the distribution tenant. Use the dropdown menu in the search bar to filter by domain, name, distribution ID, certificate ID, connection group ID, or web ACL ID.
4. Choose the distribution tenant name.
5. For **Domains**, choose **Manage domain**.
6. For **Certificate**, choose if you want a **Custom TLS certificate** for your distribution tenant. The certificate verifies whether you're authorized to use the domain name. The certificate must exist in the US East (N. Virginia) Region.
7. For **Domains**, choose **Add domain** and enter the domain name. Depending on your domain, the following messages will appear under the domain name that you enter.

- This domain is covered by the certificate.
- This domain is covered by the certificate, pending validation.
- This domain isn't covered by a certificate. (This means you must verify domain ownership.)

8. Choose **Update distribution tenant**.

   On the tenant details page, under **Domains**, you can see the following fields:

   - **Domain ownership** – The status of domain ownership. Before CloudFront can serve content, your domain ownership must be verified by using TLS certificate validation.
   - **DNS status** – Your domain's DNS records must point to CloudFront to route traffic correctly.

9. If your domain ownership isn't verified, on the tenant details page, under **Domains**, choose **Complete domain setup** and then complete the following procedure to point the DNS record to your CloudFront domain name.

Complete domain setup

Follow these procedures to verify that you own the domain for your distribution tenants. Depending on your domain, choose one of the following procedures.

Note

If your domain is already pointed to CloudFront with an Amazon Route 53 alias record, you must add your DNS TXT record with `_cf-challenge.` in front of the domain name. This TXT record verifies that your domain name is linked to CloudFront. Repeat this step for each domain. The following shows how to update your TXT record:

- Record name: `_cf-challenge.`*`DomainName`*
- Record type: TXT
- Record value: *`CloudFrontRoutingEndpoint`*

For example, your TXT record might look like: `_cf-challenge.example.com TXT d111111abcdef8.cloudfront.net`
You can find your CloudFront routing endpoint in the console on the distribution tenant detail page or use the ListConnectionGroups API action in the *Amazon CloudFront API Reference* to find it.

Tip

If you're a SaaS provider and you want to allow certificate issuance without requiring your customers (tenants) to add a TXT record directly to their DNS, do the following:

1. If you own the domain `example-saas-provider.com`, assign subdomains to your tenants, such as `customer-123.example-saas-provider.com`
2. In your DNS, add the `_cf-challenge.customer-123.example-saas-provider.com TXT d111111abcdef8.cloudfront.net` TXT record to your DNS configuration.
3. Next, your customers (the tenants) can then update their own DNS record to map their domain name to the subdomain that you provided.

   ```
   www.customer-domain.com CNAME customer-123.example-saas-provider.com
   ```

I have existing traffic

Select this option if your domain can't tolerate downtime. You must have access to your origin/web server. Use the following procedure to validate domain ownership.

To complete domain setup when you have existing traffic

1. For **Specify your web traffic**, choose **I have existing traffic** and then choose **Next**.
2. For **Verify domain ownership**, choose one of the following options:

   - **Use existing certificate** – Search for an existing ACM certificate or enter the certificate ARN that covers the listed domains.
   - **Manual file upload** – Choose if you have direct access to upload files to your web server.

     For each domain, create a plain text file that contains your validation token from the **Token location** and upload it to your origin at the specified **File path** on your existing server. The path to this file might look like the following example: `/.well-known/pki-validation/acm_9c2a7b2ec0524d09fa6013efb73ad123.txt`. After you complete that step, ACM verifies the token and then issues the TLS certificate for the domain.
   - **HTTP redirect** – Choose if you don't have direct access to upload files to your web server, or if you're using a CDN or proxy service.

For each domain, create a 301 redirect on your existing server. Copy the well-known path under **Redirect from** and point to the specified certificate endpoint under **Redirect to**. Your redirect might look like the following example:

```
If the URL matches: example.com/.well-known/pki-validation/
leabe938a4fe077b31e1ff62b781c123.txt
Then the settings are:Forwarding URL
Then 301 Permanent Redirect:To validation.us-east-1.acm-
validations.aws/123456789012/.well-known/pki-validation/
leabe938a4fe077b31e1ff62b781c123.txt
```

> **Note**
>
> You can choose **Check certificate status** to verify when ACM issues the certificate for the domain.

3. Choose **Next**.
4. Complete the steps for Point domains to CloudFront.

I don't have traffic

Select this option if you're adding new domains. CloudFront will manage certificate validation for you.

To complete domain setup if you don't have traffic

1. For **Specify your web traffic**, choose **I don't have traffic yet**.
2. For each domain name, complete the steps for Point domains to CloudFront.
3. After you update your DNS records for each domain name, choose **Next**.
4. Wait for the certificate to be issued.

> **Note**
>
> You can choose **Check certificate status** to verify when ACM issues the certificate for the domain.

5. Choose **Submit**.

Point domains to CloudFront

Update your DNS records to route traffic from each domain to the CloudFront routing endpoint. You can have multiple domain names, but they must all resolve to this endpoint.

To point domains to CloudFront

1. Copy the CloudFront routing endpoint value, such as d111111abcdef8.cloudfront.net.
2. Update your DNS records to route traffic from each domain to the CloudFront routing endpoint.
    1. Sign in to your domain registrar or DNS provider management console.
    2. Navigate to the DNS management section for your domain.
        - **For subdomains** – Create a CNAME record. For example:
            - **Name** – Your subdomain (such as `www` or `app`)
            - **Value / Target** – The CloudFront routing endpoint
            - **Record type** – CNAME
            - **TTL** – 3600 (or whatever is appropriate for your use case)
        - **For apex/root domains** – This requires a unique DNS configuration, because standard CNAME records can't be used at the root or apex domain level. Because most DNS providers don't support ALIAS records, we recommend creating an ALIAS record in Route 53. For example:
            - **Name** – Your apex domain (such as `example.com`)
            - **Record type** – A
            - **Alias** – Yes
            - **Alias target** – Your CloudFront routing endpoint
            - **Routing policy** – Simple (or whatever is appropriate for your use case)
    3. Verify that the DNS change has propagated. (This usually happens when the TTL is expired. Sometimes it can take 24-48 hours.) Use a tool like `dig` or `nslookup`.

        ```
        dig www.example.com
        # Should eventually return a CNAME pointing to your CloudFront routing endpoint
        ```
3. Return to the CloudFront console and choose **Submit**. When your domain is active, CloudFront updates the domain status to indicate that your domain is ready to serve traffic.

For more information, see the documentation for your DNS provider:

- Cloudflare
- ClouDNS
- DNSimple
- Gandi.net
- GoDaddy
- Google Cloud DNS
- Namecheap

Domain considerations (distribution tenant)

When a domain is active, domain control has been established and CloudFront will respond to all viewer requests to this domain. Once activated, a domain can't be deactivated or changed to an inactive status. The domain can't be associated with another CloudFront resource while it's already in use. To associate the domain with another distribution, use the UpdateDomainAssociation request to move the domain from one CloudFront resource to another. To move the domain to a distribution tenant, you can also set `OnDomainTakeover` to `allow` when you create or update the distribution tenant. For more information, see Move domains when you create or update a distribution tenant.

When a domain is inactive, CloudFront won't respond to viewer requests to the domain. While the domain is inactive, note the following:

- If you have a pending certificate request, CloudFront will respond to requests for the well-known path. While the request is pending, the domain can't be associated with any other CloudFront resources.
- If you don't have a pending certificate request, CloudFront won't respond to requests for the domain. You can associate the domain with other CloudFront resources.
- You can only have *one pending certificate request* per distribution tenant. Before you can request another certificate for additional domains, you must cancel the existing pending request. Canceling an existing certificate request does not delete the associated ACM certificate. You can delete that by using the ACM API.
- If you apply a new certificate to your distribution tenant, this will disassociate the previous certificate. You can reuse the certificate to cover the domain for another distribution tenant.

As with renewals for DNS-validated certificates, you will be notified when the certificate renewal succeeds. However, you don't need to do anything else. CloudFront will manage the certificate renewal for your domain automatically.

> **Note**
>
> You don't need to call the ACM API operations to create or update your certificate resources. You can manage your certificates by using the CreateDistributionTenant and UpdateDistributionTenant API operations to specify the details for your managed certificate request.

> **Note**
>
> **Geographic restrictions compatibility** — CloudFront managed certificate validation is compatible with geographic restrictions. When you enable geographic restrictions on a distribution that uses a CloudFront managed certificate, validation requests are excluded from geographic restriction rules. Certificate Authorities verify domain ownership from multiple geographically distributed network perspectives (Multi Perspective Issuance Corroboration) to protect against BGP hijacking. CloudFront makes sure these validation requests succeed regardless of which countries you block or allow.
> No action is required to use this functionality. If you previously removed geographic restrictions as a workaround for certificate validation failures, you can safely re-apply them.

Wildcard domains (distribution tenant)

Wildcard domains are supported for distribution tenants in the following situations:

- When the wildcard is included in the shared certificate that's inherited from the parent multi-tenant distribution
- When you use a valid existing custom TLS certificate for your distribution tenant

Serve distribution tenant domains without a certificate

You can add a domain to a distribution tenant even when no certificate covers it. Instead of a certificate, CloudFront uses the domain's DNS records to verify that you control the domain. CloudFront then serves the domain over HTTP. You can use this to do the following:

- Start serving HTTP traffic for a domain as soon as its DNS records point to CloudFront.
- Get a certificate for the domain from a third-party certificate authority (CA) that validates domains over HTTP, and then add that certificate to the distribution tenant. For more information, see Use a certificate from a third-party CA.

Topics

- How it works
- Use a certificate from a third-party CA

How it works

When you create or update a distribution tenant, CloudFront checks each domain that you add. If no certificate on the distribution tenant or its multi-tenant distribution covers a domain, CloudFront checks that the domain's DNS configuration points to the distribution tenant. Usually, this is the domain's CNAME record. CloudFront accepts any of the following records:

- A CNAME record that points the domain to the routing endpoint of the distribution tenant's connection group.
- An A record that resolves the domain to the Anycast static IP addresses of the distribution tenant's connection group.
- A TXT record that associates the domain with the routing endpoint of the distribution tenant's connection group.

  The name of the TXT record depends on the type of domain. The value is the CloudFront domain name of the target standard distribution, or the routing endpoint of the target distribution tenant's connection group, such as d111111abcdef8.cloudfront.net.

Domain type	Domain	TXT record
Subdomain	`www.example.com`	`_www.example.com TXT d111111abcdef8.cloudfront.net`
Apex (root) domain	`example.com`	`_.example.com TXT d111111abcdef8.cloudfront.net`
Wildcard domain	`*.example.com`	`_*.example.com TXT d111111abcdef8.cloudfront.net`

An apex domain, such as `example.com`, can't have a CNAME record, and an alias record that resolves the domain to the shared CloudFront IP addresses doesn't pass this check. If you point an apex domain to CloudFront with an alias record, also create a TXT record. The alias record routes viewers to the distribution tenant, and the TXT record passes the check. If the connection group uses Anycast static IPs, an A record that resolves the domain to those addresses does both, so you don't need the TXT record.

If the DNS check fails for a domain, the request fails with `InvalidArgument`. If you get this error when you add a domain that no certificate covers, check the domain's DNS records. A distribution tenant doesn't need any certificate if every domain passes this check.

Until a certificate covers the domain, CloudFront serves the domain over HTTP only, and HTTPS requests for the domain fail. After you add a certificate that covers the domain, CloudFront serves it over HTTPS too.

> **Important**
>
> Viewers can reach the domain over HTTP only if the cache behaviors of the multi-tenant distribution allow HTTP. If Viewer protocol policy is **Redirect HTTP to HTTPS**, CloudFront redirects viewers to HTTPS, which fails until a certificate covers the domain. If it's **HTTPS only**, HTTP requests for the domain fail.

The same rules apply to domains that you set to `anycast-ip-list-only` and to domains that you move to the distribution tenant with `OnDomainTakeover`. For more information, see Serve domains on specific IP addresses with address-based selection and Move domains when you create or update a distribution tenant.

Use a certificate from a third-party CA

Some certificate authorities validate a domain by requesting a file from it over HTTP, for example at `/.well-known/acme-challenge/` or `/.well-known/pki-validation/`. Because CloudFront serves the domain over HTTP before you have a certificate, the CA's validation request reaches your origin through CloudFront. After the CA issues the certificate, you import it into ACM and add it to the distribution tenant.

To use a certificate from a third-party CA

1. In your DNS configuration, point the domain to the distribution tenant's connection group, so that the CA's validation request reaches CloudFront. For more information, see Point domains to CloudFront.

   If you point an apex domain to CloudFront with an alias record, also create a TXT record, as described in How it works.
2. Add the domain to the distribution tenant. You don't need a certificate for it. Replace the ID, the ETag, and the domains with your own, and include every domain on the distribution tenant.

   ```
   aws cloudfront update-distribution-tenant \
       --id dt_EXAMPLE22222 \
       --if-match E3QWRUHEXAMPLE \
       --domains '[{"Domain": "www.example.com"}]'
   ```
3. Make sure that the multi-tenant distribution can serve the CA's validation request:

   - A cache behavior that matches the validation path allows HTTP and forwards the request to an origin that serves the validation file.
   - CloudFront doesn't serve an old copy of the validation file from its cache.

   For example, add a cache behavior like the following to the multi-tenant distribution. It allows HTTP, turns off caching, and forwards the `Host` header, so that your origin can return the right validation file for each domain. This example shows only the relevant settings.

```
{
    "PathPattern": "/.well-known/acme-challenge/*",
    "TargetOriginId": "my-origin",
    "ViewerProtocolPolicy": "allow-all",
    "CachePolicyId": "4135ea2d-6df8-44a3-9df3-4b5a84be39ad",
    "OriginRequestPolicyId": "216adef6-5c7f-47e4-b989-5492eafa07d3"
}
```

These IDs are the managed CachingDisabled cache policy and the managed AllViewer origin request policy.

4. Request the certificate from your CA with HTTP validation, and put the validation file on your origin as the CA instructs.
5. After the CA issues the certificate, import it into ACM in the US East (N. Virginia) (`us-east-1`) Region. For more information, see Import certificates into ACM in the *AWS Certificate Manager User Guide*.
6. Update the distribution tenant to use the certificate. Replace the ETag and the certificate ARN with your own.

```
aws cloudfront update-distribution-tenant \
    --id dt_EXAMPLE22222 \
    --if-match E4QWRUHEXAMPLE \
    --domains '[{"Domain": "www.example.com"}]' \
    --customizations '{"Certificate": {"Arn": "arn:aws:acm:us-
east-1:123456789012:certificate/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"}}'
```

After the distribution tenant is deployed, CloudFront serves the domain over HTTPS.

> **Note**
>
> ACM doesn't renew certificates that you import. Before the certificate expires, get a new certificate from your CA the same way, and reimport it to the same certificate ARN. Because the ARN doesn't change, you don't need to update the distribution tenant.

Create custom connection group (optional)

By default, CloudFront creates a connection group for you when you create a multi-tenant distribution. The connection group controls how viewer requests for content connect to CloudFront.

We recommend that you use the default connection group. However, if you need to isolate enterprise applications or manage groups of distribution tenants separately, you can choose to create a custom connection group. For example, you might need to move a distribution tenant to a separate connection group if it experiences a DDoS attack. This way, you can protect other distribution tenants from impact.

Create custom connection group (optional)

Optionally, you can choose to create a custom connection group for your distribution tenants.

To create a custom connection group (optional)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Settings**.
3. Turn on the **Connection group** settings.
4. In the navigation pane, choose **Connection groups**, and then choose **Create connection group**.
5. For **Connection group name**, enter a name for the connection group. You can't update this name after you create the connection group.
6. For **IPv6**, specify if you want to enable this IP protocol. For more information, see Enable IPv6 (viewer requests).
7. For **Anycast static IP list**, specify if you want to deliver traffic to your distribution tenants from a set of IP addresses. For more information, see Anycast static IP list.
8. (Optional) Add tags to your connection group.
9. Choose **Create connection group**.

When your connection group is created, you can find the settings that you specified, and also the ARN and endpoint.

- The ARN looks like the following example: `arn:aws:cloudfront::123456789012:connection-group/cg_2uVbA9KeWaADTbKzhj9lcKDoM25`
- The endpoint looks like the following example: d111111abcdef8.cloudfront.net

You can edit or delete your custom connection group after you create it. Before you can delete a connection group, you must first delete all associated distribution tenants from it. You can't disable or delete the default connection group that CloudFront created for you when you created your multi-tenant distribution.

If your connection group uses an Anycast static IP list, you can also choose which domains its distribution tenants serve only on that list, how CloudFront handles viewers that don't send SNI, and get connection logs for HTTPS connections to its distribution tenants. For more information, see Serve domains on specific IP addresses with address-based selection, Serve viewers that don't send SNI with a viewer fallback, and Connection logs for distribution tenants on Anycast static IPs.

Important

If you change the connection group for a distribution tenant, CloudFront will continue to carry traffic for the distribution tenant, but with increased latency. We recommend that you update the DNS record for the distribution tenant to use the CloudFront routing endpoint from the new connection group.
Until you update the DNS record, CloudFront will route based on settings defined for the routing endpoint that the website is currently pointing to with DNS. For example, assume that your default connection group does not use Anycast static IPs but your new, custom connection group does. You must update the DNS record before CloudFront will use Anycast static IPs for the distribution tenants in your custom connection group.

Serve domains on specific IP addresses with address-based selection

With address-based selection, CloudFront selects the distribution tenant for a viewer request based on both the domain name and the IP address that the request arrives on. You can use address-based selection to do the following:

- Serve the same wildcard domain, such as `*.example.com`, from more than one distribution tenant, each on its own Anycast static IP list. Your DNS records decide which IP addresses a hostname resolves to, and so which distribution tenant serves it.
- Serve a wildcard domain only on your own Anycast static IP addresses, so that CloudFront doesn't serve it on other IP addresses.

You can use address-based selection only with distribution tenants, in connection groups that use an Anycast static IP list. For more information, see Create custom connection group (optional) and Request Anycast static IPs to use for allowlisting.

Topics

- How address-based selection works
- Use cases
- Address-based selection requirements and restrictions
- Required permissions
- Set up address-based selection
- Example: Serve three service tiers under one wildcard domain
- Troubleshoot address-based selection

How address-based selection works

A distribution tenant gets its IP addresses from its connection group. For each domain on a distribution tenant, the `ServeOn` setting controls which IP addresses the distribution tenant serves that domain on:

- `any` (default) – The distribution tenant serves the domain on any CloudFront IP address. This includes the shared CloudFront IP addresses and your Anycast static IP lists. This is how CloudFront serves domains when you don't use address-based selection.
- `anycast-ip-list-only` – The distribution tenant serves the domain only on the IP addresses of the Anycast static IP list that's associated with its connection group.

CloudFront selects the resource that serves a request based on the IP address that the request arrives on. When a request arrives on the IP addresses of an Anycast static IP list, CloudFront selects the resource in the following order:

1. A standard distribution or distribution tenant whose alternate domain name exactly matches the requested domain.
2. The distribution tenant that serves a matching wildcard domain with `anycast-ip-list-only` on that Anycast static IP list. If more than one wildcard domain matches, CloudFront selects the most specific one.
3. A standard distribution or distribution tenant whose wildcard alternate domain name matches the requested domain.

For requests that arrive on any other IP address, CloudFront ignores domains that are set to `anycast-ip-list-only`. If no resource serves the requested domain on the IP address that the request arrived on, CloudFront rejects the request.

> **Important**
>
> When you set a domain that already receives traffic to `anycast-ip-list-only`, the distribution tenant stops serving it on every other IP address. Before you update the distribution tenant, point the DNS record for every hostname under the domain to the connection group's routing endpoint or Anycast static IP addresses.

Use cases

Address-based selection supports the following use cases:

Service tiers

Serve paid tiers from dedicated IP addresses, each with its own configuration, and serve a free tier on the shared CloudFront IP addresses, all under the same wildcard domain.

Sharding

Split a large customer base across several distribution tenants, each with its own IP addresses, origins, and configuration. To move a customer to a different shard, you change only that customer's DNS record.

Address-based selection requirements and restrictions

To serve a domain with `anycast-ip-list-only`, the following must be true:

- The distribution tenant's connection group uses an Anycast static IP list, and no other connection group uses that list.
- The domain is a wildcard domain with one or two labels before the top-level domain, such as `*.example.com` or `*.app.example.com`. You can't use an exact domain name, such as `www.example.com` or `example.com`, or a wildcard of a public suffix, such as `*.co.uk`.
- A certificate that you provide covers the domain, or a DNS record for the domain points to the connection group. The certificate can be the distribution tenant's own certificate or its multi-tenant distribution's certificate. Without a certificate, CloudFront verifies the DNS record instead and serves the domain over HTTP only. For more information, see Serve distribution tenant domains without a certificate. `ServeOn` doesn't change the domain control and certificate requirements for a domain. For more information, see Domain considerations (distribution tenant).

> **Note**
>
> CloudFront managed certificates don't support domains that are set to `anycast-ip-list-only`. CloudFront doesn't include these domains in a managed certificate request, so a managed certificate on the distribution tenant doesn't cover them.

Note the following restrictions:

- A distribution tenant can serve up to 10 domains with `anycast-ip-list-only`.
- On a given Anycast static IP list, only one distribution tenant can serve a domain with `anycast-ip-list-only`. You can serve the same domain from other distribution tenants that use other Anycast static IP lists.
- A distribution tenant can't serve the same domain with both `any` and `anycast-ip-list-only`.
- The rule that an alternate domain name can be associated with only one CloudFront resource doesn't apply to domains that are set to `anycast-ip-list-only`. Another standard distribution or distribution tenant can use the same domain as an alternate domain name.
- `anycast-ip-list-only` applies to every IP address in the Anycast static IP list. You can't serve a domain on only some of the IP addresses in a list.
- When a distribution tenant in the connection group serves a domain with `anycast-ip-list-only`, you can't change the connection group's Anycast static IP list or IPv6 setting, change the Anycast static IP list itself, such as its IP address type, or associate the Anycast static IP list with another connection group. To make these changes, first remove the `anycast-ip-list-only`

domains from the distribution tenants in the connection group, and remove the connection group's viewer fallback if it has one. For more information, see Viewer fallback requirements and restrictions.

- To move a distribution tenant that serves domains with `anycast-ip-list-only` to another connection group, the new connection group must also use its own Anycast static IP list, and no other distribution tenant can already serve the domains on that list. After the move, the distribution tenant serves the domains on the new list. You can't move the distribution tenant to a connection group that doesn't use an Anycast static IP list, such as the default connection group.
- To move a domain that's set to `anycast-ip-list-only` from one distribution tenant to another distribution tenant on the same Anycast static IP list, first update the source distribution tenant to remove the domain, and then update the target distribution tenant to add it. Between the two updates, no distribution tenant serves the domain on that list.

Required permissions

To set up address-based selection, you need permission to create connection groups and to create and update distribution tenants. The following policy grants the permissions for the procedure in this topic.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "cloudfront:CreateConnectionGroup",
            "Resource": "arn:aws:cloudfront::123456789012:connection-group/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudfront:CreateDistributionTenant",
                "cloudfront:UpdateDistributionTenant",
                "cloudfront:GetDistributionTenant"
            ],
            "Resource": "arn:aws:cloudfront::123456789012:distribution-tenant/*"
        }
    ]
}
```

Set up address-based selection

The following procedure uses the AWS CLI to serve `*.example.com` from a distribution tenant only on the IP addresses of an Anycast static IP list. Before you begin, you need a multi-tenant distribution, an Anycast static IP list, and an ACM certificate in the US East (N. Virginia) (`us-east-1`) Region that covers the domain.

To set up address-based selection

1. Create a connection group that uses your Anycast static IP list. Replace the Anycast static IP list ID with your own.

```
aws cloudfront create-connection-group \
    --name example-dedicated-group \
    --anycast-ip-list-id aip_EXAMPLE11111
```

   In the response, note the connection group `Id` and `RoutingEndpoint`. The connection group's `Status` is `InProgress` until CloudFront deploys it, which can take a few minutes.

```
{
    "ETag": "E2QWRUHEXAMPLE",
    "ConnectionGroup": {
        "Id": "cg_EXAMPLE11111",
        "Name": "example-dedicated-group",
        "Arn": "arn:aws:cloudfront::123456789012:connection-group/cg_EXAMPLE11111",
        "RoutingEndpoint": "d111111abcdef8.cloudfront.net",
        "AnycastIpListId": "aip_EXAMPLE11111",
        "Ipv6Enabled": false,
        "Status": "InProgress",
        "Enabled": true,
        "IsDefault": false
    }
}
```

2. Create a distribution tenant in the connection group, and set `ServeOn` to `anycast-ip-list-only` for the domain. Replace the multi-tenant distribution ID, the connection group ID, and the certificate ARN with your own.

```
aws cloudfront create-distribution-tenant \
    --distribution-id EDFDVBD6EXAMPLE \
    --name example-dedicated \
```

```
    --connection-group-id cg_EXAMPLE11111 \
    --domains '[{"Domain": "*.example.com", "ServeOn": "anycast-ip-list-only"}]' \
    --customizations '{"Certificate": {"Arn": "arn:aws:acm:us-
east-1:123456789012:certificate/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"}}'
```

The `Domains` list in the response shows the `ServeOn` value for each domain.

```
{
    "ETag": "E3QWRUHEXAMPLE",
    "DistributionTenant": {
        "Id": "dt_EXAMPLE22222",
        "DistributionId": "EDFDVBD6EXAMPLE",
        "Name": "example-dedicated",
        "Domains": [
            {
                "Domain": "*.example.com",
                "ServeOn": "anycast-ip-list-only",
                "Status": "active"
            }
        ],
        "ConnectionGroupId": "cg_EXAMPLE11111",
        "Enabled": true,
        "Status": "InProgress"
    }
}
```

To change an existing distribution tenant instead, use the **update-distribution-tenant** command with the distribution tenant's `--id`, its current ETag in `--if-match`, and the same `--domains` and `--connection-group-id` values. Include every domain on the distribution tenant in `--domains`.

3. In your DNS configuration, create a CNAME record for each hostname that points to the connection group's routing endpoint. The routing endpoint resolves to the IP addresses of the connection group's Anycast static IP list.

```
app1.example.com.  300  IN  CNAME  d111111abcdef8.cloudfront.net.
```

4. After the distribution tenant's status changes to `Deployed`, confirm that the hostname resolves to an IP address on your Anycast static IP list, and that CloudFront serves the request.

```
dig +short app1.example.com
```

```
curl -sI https://app1.example.com
```

Example: Serve three service tiers under one wildcard domain

A software-as-a-service (SaaS) provider hosts customer apps under `*.example.com` and offers premium, pro, and free tiers. Premium and pro customers get dedicated IP addresses and their own configuration, and free customers share CloudFront's IP addresses. The provider creates three distribution tenants that all serve `*.example.com`. The following table shows each distribution tenant's connection group and `ServeOn` setting, and the IP addresses that it serves requests on.

Distribution tenant	Connection group	`ServeOn`	Serves requests that arrive on
premium	Uses the premium Anycast static IP list	`anycast-ip-list-only`	The premium list
pro	Uses the pro Anycast static IP list	`anycast-ip-list-only`	The pro list
free	Default connection group	`any`	Every other CloudFront IP address

Each customer's hostname is a CNAME record to the routing endpoint of the connection group for that customer's tier. A request for `customer1.example.com` that resolves to the premium list goes to the premium distribution tenant. A request for `customer2.example.com` that resolves to a shared CloudFront IP address goes to the free distribution tenant. To move a customer to another tier, the provider changes only that customer's DNS record.

> **Note**
>
> Because the free distribution tenant serves `*.example.com` on any IP address, it also receives requests for premium or pro hostnames whose DNS records point to the shared CloudFront IP addresses, for example because of an incorrect record. Those requests get the free tier's configuration instead of being rejected. If you want such requests to fail, serve every tier on its own Anycast static IP list.

Troubleshoot address-based selection

The following are common issues with address-based selection and how to resolve them.

CloudFront rejects requests for the domain

The hostname resolves to an IP address that no distribution tenant serves the domain on. Use **dig** to see which IP addresses the hostname resolves to, and check that its DNS record points to the routing endpoint of the connection group whose distribution tenant serves the domain.

The wrong distribution tenant serves requests

CloudFront selects the distribution tenant from the IP address that the request arrives on, so check the hostname's DNS record first. If the record points to the shared CloudFront IP addresses, a resource that serves the domain with `any` receives the request. If another standard distribution or distribution tenant has the exact hostname as an alternate domain name, that resource receives the request.

The **get-distribution-tenant-by-domain** command doesn't find distribution tenants by domains that are set to `anycast-ip-list-only`. It returns only a distribution tenant that has the domain as a regular alternate domain name. To see which distribution tenant serves a domain on an Anycast static IP list, list the distribution tenants in the connection group and check their `Domains`:

```
aws cloudfront list-distribution-tenants \
    --association-filter ConnectionGroupId=cg_EXAMPLE11111
```

Creating or updating a distribution tenant fails

Check the following:

- Another distribution tenant already serves the domain with `anycast-ip-list-only` on the same Anycast static IP list.
- The connection group doesn't use an Anycast static IP list, or another connection group also uses the list.
- The domain isn't a supported wildcard domain. Or no certificate on the distribution tenant or its multi-tenant distribution covers the domain, and CloudFront couldn't verify the domain's DNS records instead. A CloudFront managed certificate doesn't cover domains that are set to `anycast-ip-list-only`.

Serve viewers that don't send SNI with a viewer fallback

Some HTTPS clients, such as older libraries, embedded devices, and some proxies, don't send Server Name Indication (SNI) in the TLS handshake. Without SNI, CloudFront can't tell which certificate to present, so it rejects these handshakes.

With a viewer fallback, you choose a multi-tenant distribution whose certificate CloudFront uses for viewers that connect to your connection group's Anycast static IP addresses without SNI. You set the viewer fallback on the connection group.

Topics

- How viewer fallback works
- Viewer fallback requirements and restrictions
- Set a viewer fallback

How viewer fallback works

When a viewer connects to one of the connection group's Anycast static IP addresses and doesn't send SNI, CloudFront completes the TLS handshake with the certificate and TLS settings of the viewer fallback multi-tenant distribution. CloudFront then routes each request on the connection by its `Host` header, as usual.

The viewer fallback multi-tenant distribution only supplies TLS settings for the handshake. CloudFront doesn't route requests to it, and it doesn't need to serve any domain on the Anycast static IP list. Viewers that send SNI aren't affected by the viewer fallback.

When the certificate on the viewer fallback multi-tenant distribution changes or renews, viewers without SNI get the new certificate. You don't need to update the connection group.

Example – API clients that don't send SNI

A connection group that uses an Anycast static IP list contains distribution tenants for an API at `api.example.com` and for the sign-in domains of several customers. Some of the API's clients don't send SNI. The multi-tenant distribution `E2EXAMPLE33333` has a certificate for `*.example.com`, so you set it as the connection group's viewer fallback.

The following table shows how CloudFront handles requests on the connection group's Anycast static IP addresses.

SNI	Host	Result
`login.customer.com`	`login.customer.com`	The customer's distribution tenant serves the request. The viewer fallback doesn't apply.
None	`api.example.com`	CloudFront completes the handshake with the `*.example.com` certificate from `E2EXAMPLE33333`. The API distribution tenant serves the request.
None	`login.customer.com`	CloudFront completes the handshake with the `*.example.com` certificate from `E2EXAMPLE33333`, which doesn't cover `login.customer.com`. A viewer that validates the certificate ends the connection.

Viewer fallback requirements and restrictions

To set a viewer fallback, the following must be true:

- The connection group uses an Anycast static IP list, and no other connection group uses that list.
- The viewer fallback is a multi-tenant distribution in the same AWS account as the connection group. You can't use a standard distribution.
- The multi-tenant distribution is enabled and has a custom TLS certificate, not the default `*.cloudfront.net` certificate.

When a connection group has a viewer fallback, the following restrictions apply:

- You can't change the connection group's Anycast static IP list or IPv6 setting, unless you remove the viewer fallback in the same update.
- You can't change the Anycast static IP list itself, such as its IP address type. To change the list, first remove the viewer fallback from the connection group.
- Removing the viewer fallback doesn't unblock these changes if a distribution tenant in the connection group also serves a domain with `anycast-ip-list-only`. Remove those domains too. For more information, see Address-based selection requirements and restrictions.
- You can't associate the Anycast static IP list with another connection group.
- You can't disable or delete the viewer fallback multi-tenant distribution, or replace its custom certificate with the default `*.cloudfront.net` certificate.

Set a viewer fallback

You can set a viewer fallback when you create a connection group with the **create-connection-group** command, or on an existing connection group with the **update-connection-group** command. The following example sets a viewer fallback on an existing connection group.

To set a viewer fallback

1. Get the connection group's current ETag. Replace the connection group ID with your own.

```
aws cloudfront get-connection-group \
    --identifier cg_EXAMPLE33333
```

2. Update the connection group. Replace the connection group ID, the ETag, and the multi-tenant distribution ID with your own.

```
aws cloudfront update-connection-group \
    --id cg_EXAMPLE33333 \
    --if-match E1ABCDEXAMPLE \
    --viewer-fallback DistributionId=E2EXAMPLE33333
```

   The response shows the viewer fallback. The connection group's `Status` is `InProgress` until CloudFront deploys the change, which can take a few minutes.

```
{
    "ETag": "E2QWRUHEXAMPLE",
    "ConnectionGroup": {
```

```
        "Id": "cg_EXAMPLE33333",
        "Name": "example-group",
        "Arn": "arn:aws:cloudfront::123456789012:connection-group/cg_EXAMPLE33333",
        "RoutingEndpoint": "d111111abcdef8.cloudfront.net",
        "AnycastIpListId": "aip_EXAMPLE11111",
        "ViewerFallback": {
            "DistributionId": "E2EXAMPLE33333"
        },
        "Ipv6Enabled": false,
        "Enabled": true,
        "Status": "InProgress",
        "IsDefault": false
    }
}
```

If you omit `--viewer-fallback` when you update the connection group, CloudFront keeps the current viewer fallback. To remove the viewer fallback, specify an empty viewer fallback. Replace the connection group ID and the ETag with your own.

```
aws cloudfront update-connection-group \
    --id cg_EXAMPLE33333 \
    --if-match E2QWRUHEXAMPLE \
    --viewer-fallback '{}'
```

Migrate to a multi-tenant distribution

If you have a CloudFront standard distribution and you want to migrate to a multi-tenant distribution, follow these steps.

To migrate from a standard distribution to a multi-tenant distribution

1. Review the Unsupported features.
2. Create a multi-tenant distribution with the same configuration as your standard distribution, minus the unsupported features. For more information, see Create a CloudFront distribution in the console.
3. Create a distribution tenant and add an alternative domain name that you own.

> **Warning**
>
> Do *not* use the current domain name that's associated with your standard distribution. Add a placeholder domain instead. You will move your domain over later. For more information about creating a distribution tenant, see Create a CloudFront distribution in the console.

4. Provide an existing certificate for the distribution tenant domain. This is the certificate that will cover the placeholder domain and the domain that you want to move.
5. Copy the CloudFront routing endpoint from the distribution tenant detail page in the console. Alternatively, find it by using the ListConnectionGroups API action in the *Amazon CloudFront API Reference*.
6. To verify domain ownership, create a DCV TXT record with an underscore (_) prefix that points to the CloudFront routing endpoint for your distribution tenant. For more information, see Point domains to CloudFront.
7. When your changes have propagated, update your distribution tenant to use the domain that you previously used for your standard distribution.

   - **Console** – For detailed instructions, see Add a domain and certificate (distribution tenant).
   - **API** – Use the UpdateDomainAssociation API action in the *Amazon CloudFront API Reference*.

> **Important**
>
> This resets the cache key for your content. After this, CloudFront starts caching your content using the new cache key. For more information, see Understand the cache key.

8. Update your DNS record to point your domain to the CloudFront routing endpoint for your distribution tenant. After you complete this step, your domain will be ready to serve traffic to your distribution tenant. For more information, see Point domains to CloudFront.
9. (Optional) After you successfully migrate your domain to a distribution tenant, you can use a different CloudFront managed certificate that covers the domain name for your distribution tenant. To request a managed certificate, create a separate TXT record to issue the certificate and follow the steps here in Complete domain setup.

Create a distribution

This topic explains how to use the CloudFront console to create a distribution.

Overview

1. Create one or more Amazon S3 buckets, or configure HTTP servers as your origin servers. An *origin* is the location where you store the original version of your content. When CloudFront gets a request for your files, it goes to the origin to get the files that it distributes at edge locations. You can use any combination of Amazon S3 buckets and HTTP servers as your origin servers.

   - If you use Amazon S3, the name of your bucket must be all lowercase and cannot contain spaces.
   - If you use an Amazon EC2 server or another custom origin, review Use Amazon EC2 (or another custom origin).
   - For the current maximum number of origins that you can create for a distribution, or to request a higher quota, see General quotas on distributions.
2. Upload your content to your origin servers. You make your objects publicly readable, or you can use CloudFront signed URLs to restrict access to your content.

   > **Important**
   >
   > You are responsible for making sure the security of your origin server. You must make sure that CloudFront has permission to access the server and that the security settings safeguard your content.

3. Create your CloudFront distribution:

   - For a detailed procedure that creates a distribution in the CloudFront console, see Create a CloudFront distribution in the console.
   - For information about creating a distribution using the CloudFront API, see CreateDistribution in the *Amazon CloudFront API Reference*.
4. (Optional) If you use the CloudFront console to create your distribution, create more cache behaviors or origins for the distribution. For more information about behaviors and origins, see To update a multi-tenant distribution.
5. Test your distribution. For more information about testing, see Test a distribution.

6. Develop your website or application to access your content using the domain name that CloudFront returned after you created your distribution in Step 3. For example, if CloudFront returns d111111abcdef8.cloudfront.net as the domain name for your distribution, the URL for the file `image.jpg` in an Amazon S3 bucket or in the root directory on an HTTP server is `https://d111111abcdef8.cloudfront.net/image.jpg`.

   If you specified one or more alternate domain names (CNAMEs) when you created your distribution, you can use your own domain name. In that case, the URL for `image.jpg` might be `https://www.example.com/image.jpg`.

   Note the following:

   - If you want to use signed URLs to restrict access to your content, see Serve private content with signed URLs and signed cookies.
   - If you want to serve compressed content, see Serve compressed files.
   - For information about CloudFront request and response behavior for Amazon S3 and custom origins, see Request and response behavior.

Topics

- Create a CloudFront distribution in the console
- Values that CloudFront displays in the console
- Additional links
- Add a domain to your CloudFront standard distribution

Create a CloudFront distribution in the console

When you create a distribution, CloudFront configures your distribution settings for you, based on your content origin type. For more details about the preconfigured settings, see Preconfigured distribution settings reference. You can also create multi-tenant distributions with settings that can be reused across multiple distribution tenants. For more information, see Understand how multi-tenant distributions work. Alternatively, you can manually configure your own distribution settings.

Multi-tenant

To create a multi-tenant distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, then choose **Create distribution**.
3. Choose **Multi-tenant architecture**, **Next**.
4. Enter a **Distribution name** for the multi-tenant distribution. The name will appear as the value for the `Name` key. You can change this value later. You can add up to 50 tags for your multi-tenant distribution. For more information, see Tag a CloudFront resource .
5. (Optional) For **Wildcard certificate**, choose the AWS Certificate Manager (ACM) certificate that will cover all subdomains under the root domain, such as *`*.example.com`*. The certificate must be in the US East (N. Virginia) Region.
6. Choose **Next**.
7. On the **Specify origin** page, select the origin type that CloudFront will get your content from. CloudFront will use the recommended settings for that origin type for your multi-tenant distribution. For more information about the recommended settings, see Preconfigured distribution settings reference.
8. For **Origin**, under the origin type that you selected, choose or enter the origin to use.
9. For **Origin path**, enter the forward slash (*/*) character, followed by the origin path.
10. (Optional) To add a parameter, choose **Insert parameter** for either the origin domain name or origin path. You can enter up to two parameters for each field.

    a. Choose **Create new parameter**.

    b. On the **Create new parameter** dialog box, for **Parameter name**, enter a unique name for the parameter and, optionally, a description.

    c. For **Required parameter**, select the checkbox to make this parameter value required at the distribution tenant level. If it's not required, enter a **Default value** that the distribution tenant will inherit.

    d. Choose **Create parameter**. This parameter appears in the corresponding field.
11. For **Options**, choose one of the following options:

    - **Use recommended origin settings** – Use the default recommended cache and origin settings for the origin type that you selected.

- **Customize origin settings** – Customize the cache and origin settings. If you choose this option, specify your own values that appear.

12. Choose **Next**.
13. On the **Enable security protections** page, choose whether to enable AWS WAF security protections. You can customize the web ACL for specific distribution tenants later. For more information, see Enable AWS WAF for a new distribution.
14. Choose **Next**, **Create distribution**.
15. On the **Distributions** page, your multi-tenant distribution appears in the list of resources. You can choose the **All distributions** dropdown to filter by standard distribution or multi-tenant distribution. You can also choose the **Type** column to filter by standard or multi-tenant distribution.

By default, CloudFront creates a connection group for you. The connection group controls how viewer requests for content connect to CloudFront. You can customize some routing settings in the connection group. For more information, see Understand how multi-tenant distributions work.

You can create additional distribution tenants using the multi-tenant distribution as a template.

To create a distribution tenant

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, do one of the following:
   - Choose **Distributions**, choose a multi-tenant distribution, and then choose **Create tenant**.
   - Choose **Distribution tenants** and then choose **Create tenant**.
3. For **Distribution tenant name**, enter the name. The name must be unique in your AWS account and can't be changed after you create it.
4. For **Template distribution**, choose a multi-tenant distribution ID from the list.
5. For **Manage tags**, add up to 50 key-value pairs for the distribution tenant. For more information, see Tag a CloudFront resource.
6. Choose **Next**.

7. On the **Add domains** page, for **Certificate**, choose if you want a **Custom TLS certificate** for your distribution tenant. The certificate verifies whether you're authorized to use the domain name. The certificate must exist in the US East (N. Virginia) Region.
8. For **Domains**, enter your domain name.

> **Note**
>
> If you entered a domain name that isn't covered by a certificate, you will need to verify that you own the domain. You can still create the distribution tenant for now and verify domain ownership later. For more information, see Request certificates for your CloudFront distribution tenant.

9. Choose **Next**.
10. On the **Define parameters** page, the parameters that you specified in the multi-tenant distribution appear. For required parameters, enter a value next to the parameter name and save your changes.
11. To add another parameter, choose **Add parameter** and enter a name and value.
12. Choose **Next**.
13. (Optional) For **Security customization**, if you choose to **Override distribution settings**, select the option for your use case.
14. (Optional) For **Geographic restrictions customization**, if you choose to **Override distribution settings**, select the appropriate **Restriction type** and **Countries** for the distribution tenant. For more information, see Restrict the geographic distribution of your content.
15. Choose **Next**.
16. Choose **Create distribution tenant**.

You can find all your distribution tenants on the **Distribution tenants** page. You can filter by the following:

Association

- Distribution ID
- Certificate ID
- Connection group ID

- Web ACL ID

Properties

- Name
- Domain

You can edit your distribution tenants to customize specific settings. For more information, see Distribution tenant customizations.

Standard

To create a standard distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, then choose **Create distribution**.
3. Enter a **Distribution name** for the standard distribution. The name will appear as the value for the `Name` key as a tag. You can change this value later. You can add up to 50 tags for your standard distribution. For more information, see Tag a CloudFront resource.
4. Choose **Single website or app**, **Next**.
5. (Optional) For **Domain setup**, enter a domain that's already registered with Route 53 in your AWS account, or register a new domain. Complete the setup steps.
   - If your domain uses a DNS provider other than Route 53, you can still add the domain, but you'll need to do so after creating the distribution. Skip the domain setup for now to proceed with distribution creation. You must manually configure the domain and TLS certificate later. For more information, see Add a domain to your CloudFront standard distribution.
6. Choose **Next**.
7. On the **Specify origin** page, select the origin type that CloudFront will get your content from. CloudFront will use the recommended settings for that origin type for your distribution. For more information about the recommended settings, see Preconfigured distribution settings reference.
8. For **Origin**, choose or enter your origin.
9. For **Settings**, choose one of the following options:

- **Use recommended origin settings** – Use the default recommended cache and origin settings for the origin type that you selected.
- **Customize origin settings** – Customize the cache and origin settings. If you choose this option, specify your own values.

10. Choose **Next**.
11. On the **Enable security protections** page, choose whether to enable AWS WAF security protections.
12. Choose **Next**.
13. (Optional) If you are using Route 53 for your domain, you will see the **TLS certificate** page. If CloudFront can't find an existing AWS Certificate Manager (ACM) certificate for your domain in your AWS account in the `us-east-1` AWS Region, you can choose to automatically create a certificate or manually create it. After the certificate is created, choose **Next**.
14. Review your distribution details and choose **Create distribution**.
15. After CloudFront creates your distribution, the value of the **Status** column for your distribution will change from **Deploying** to the date and time that the distribution is deployed.

    The domain name that CloudFront assigns to your distribution appears in the list of distributions. (It also appears on the **General** tab for a selected distribution.)

    > **Tip**
    >
    > You can use an alternate domain name, instead of the name assigned to you by CloudFront, by following the steps in Use custom URLs by adding alternate domain names (CNAMEs).

16. When your distribution is deployed, confirm that you can access your content by using your new CloudFront URL (d111111abcdef8.cloudfront.net) or the CNAME. For more information, see Test a distribution.
17. Make sure to update your DNS records to point to CloudFront when you're ready to send traffic to your distribution. For more information, see Point domains to CloudFront (standard distribution).

Values that CloudFront displays in the console

When you create a new distribution or update an existing distribution, CloudFront displays the following information in the CloudFront console.

> **Note**
>
> Active trusted signers, the AWS accounts that have an active CloudFront key pair and can be used to create valid signed URLs, are currently not visible in the CloudFront console.

Distribution ID

When you perform an action on a distribution using the CloudFront API, you use the distribution ID to specify which distribution to use, for example, `EDFDVBD6EXAMPLE`. You can't change a distribution's distribution ID.

Deploying and status

When you deploy a distribution, you see the **Deploying** status under the **Last modified** column. Wait for the distribution to finish deploying and make sure the **Status** column shows **Enabled**. For more information, see Distribution state.

Last modified

The date and time that the distribution was last modified, using ISO 8601 format, for example, 2012-05-19T19:37:58Z. For more information, see https://www.w3.org/TR/NOTE-datetime.

Domain name

You use the distribution's domain name in the links to your objects. For example, if your distribution's domain name is `d111111abcdef8.cloudfront.net`, the link to `/images/image.jpg` would be `https://d111111abcdef8.cloudfront.net/images/image.jpg`. You can't change the CloudFront domain name for your distribution. For more information about CloudFront URLs for links to your objects, see Customize the URL format for files in CloudFront.

If you specified one or more alternate domain names (CNAMEs), you can use your own domain names for links to your objects instead of using the CloudFront domain name. For more information about CNAMEs, see Alternate domain names (CNAMEs).

> **Note**
>
> CloudFront domain names are unique. Your distribution's domain name was never used for a previous distribution and will never be reused for another distribution in the future.

Additional links

For more information about creating a distribution, see the following links.

- To learn how to create a distribution that uses an Amazon Simple Storage Service (Amazon S3) bucket origin with origin access control (OAC), see Get started with a CloudFront standard distribution.
- For information about using the CloudFront APIs to create a distribution, see CreateDistribution in the *Amazon CloudFront API Reference*.
- For information about updating a distribution (for example, to add cache behaviors to standard distributions, or to customize distribution tenants), see Update a distribution.
- To see the current maximum number of distributions that you can create for each AWS account, or to request a higher quota (formerly known as limit), see General quotas on distributions.

Add a domain to your CloudFront standard distribution

After you create a new CloudFront standard distribution, you can add a domain to it. Optionally, you can set up an Amazon Route 53 domain for your standard distribution when you create it. For more information, see Create a CloudFront distribution in the console.

Add a domain to your existing standard distribution

To add a domain to your standard distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, then choose the distribution ID.
3. Under **Settings**, **Alternate domain names**, choose **Add a domain**.
4. Enter up to five domains to serve.
5. Choose **Next**.

6. For **TLS certificate**, if CloudFront can't find an existing AWS Certificate Manager (ACM) certificate for your domain in your AWS account in the `us-east-1` AWS Region, you can create one.

   - If you are using Amazon Route 53 (Route 53), CloudFront automatically creates a certificate for you.

7. When your certificate is provisioned, you must update your DNS records with your DNS provider to prove domain ownership. Then, choose **Validate certificate**. For more information, see Point domains to CloudFront (standard distribution).

   - If you are using Route 53, CloudFront updates your DNS records for you.

8. Choose **Next**.
9. Review your changes and choose **Add domains**.
10. Before you send traffic to your distribution, make sure to update your DNS records to point to CloudFront. For more information, choose **Route domains to CloudFront** in the **Settings** section of your distribution details page.

    - If you are using Route 53, you can have CloudFront set up DNS routing for you automatically.

Point domains to CloudFront (standard distribution)

Update your DNS records to route traffic from each domain to the CloudFront hostname. You can have multiple domain names, but they must all resolve to this hostname.

To point domains to CloudFront

1. Copy the CloudFront hostname value, such as d111111abcdef8.cloudfront.net.
2. Update your DNS records to route traffic from each domain to the CloudFront hostname.

   1. Sign in to your domain registrar or DNS provider management console.
   2. Navigate to the DNS management section for your domain.

      - **For subdomains** – Create a CNAME record. For example:
        - **Name** – Your subdomain (such as `www` or `app`)
        - **Value / Target** – Your CloudFront hostname
        - **Record type** – CNAME

- **TTL** – 3600 (or whatever is appropriate for your use case)

- **For apex/root domains** – This requires a unique DNS configuration, because standard CNAME records can't be used at the root or apex domain level. Because most DNS providers don't support ALIAS records, we recommend creating an ALIAS record in Route 53. For example:
  - **Name** – Your apex domain (such as `example.com`)
  - **Record type** – A
  - **Alias** – Yes
  - **Alias target** – Your CloudFront hostname
  - **Routing policy** – Simple (or whatever is appropriate for your use case)

3. Verify that the DNS change has propagated. (This usually happens when the TTL is expired. Sometimes it can take 24-48 hours.) Use a tool like `dig` or `nslookup`.

```
dig www.example.com
# Should eventually return a CNAME pointing to your CloudFront hostname
```

3. Return to the CloudFront console and choose **Submit**. When your domain is active, CloudFront updates the domain status to indicate that your domain is ready to serve traffic.

For more information, see the documentation for your DNS provider:

- Cloudflare
- ClouDNS
- DNSimple
- Gandi.net
- GoDaddy
- Google Cloud DNS
- Namecheap

Preconfigured distribution settings reference

When you create your CloudFront distribution, CloudFront automatically configures most distribution settings for you, based on your content origin type. Optionally, you can choose

to manually edit your distribution settings. For more information, see All distribution settings reference.

The following sections describe the default preconfiguration settings for distributions, and the settings that you can customize.

Amazon S3 origin

Following are the origin settings that CloudFront preconfigures for your Amazon S3 origin in a multi-tenant distribution.

Origin settings (preconfigured)

- **Origin Access Control (console only)** – CloudFront sets this up for you. CloudFront attempts to add the S3 bucket policy for standard distributions and for multi-tenant distributions with no parameters used in the origin domain.
- **Add custom header** – None
- **Enable Origin Shield** – No
- **Connection attempts** – 3

Following are the cache settings that CloudFront preconfigures for your Amazon S3 origin in a multi-tenant distribution.

Cache settings (preconfigured)

- **Compress objects automatically** – Yes
- **Viewer protocol policy** – Redirect to HTTPS
- **Allowed HTTP method** – `GET, HEAD`
- **Restrict viewer access** – No
- **Cache policy** – `CachingOptimized`
- **Origin request policy** – None
- **Response header policy** – None
- **Smooth Streaming** – No
- **Field level encryption** – No
- **Enable real-time access logs** – No
- **Functions** – No

Following are the settings that you can customize for your Amazon S3 origin in a multi-tenant distribution.

Customizable settings

- **S3 access** – CloudFront sets this for you, based on your S3 bucket settings:
  - **If your bucket is public** – No Origin Access Control (OAC) policy is needed.
  - **If your bucket is private** – You can choose or create an OAC policy to use.
- **Enable Origin Shield** – No
- **Compress objects automatically** – Yes
  - If you choose **Yes**, then the `CachingOptimized` caching policy is used.
  - If you choose **No**, then the `CachingOptimizedForUncompressedObjects` caching policy is used.

API Gateway origin

Following are the origin settings that CloudFront preconfigures for your API Gateway origin in a multi-tenant distribution.

Origin settings (preconfigured)

- **Protocol** – HTTPS only
- **HTTPS port** – 443
- **Minimum origin SSL protocol** – TLSv1.2
- **Origin path** – None
- **Origin Access Control (console only)** – CloudFront sets this up for you
- **Add custom header** – None
- **Enable Origin Shield** – No
- **Connection attempts** – 3
- **Response timeout** – 30
- **Keep-alive timeout** – 5

Following are the cache settings that CloudFront preconfigures for your API Gateway origin in a multi-tenant distribution.

Cache settings (preconfigured)

- **Compress objects automatically** – Yes
- **Viewer protocol policy** – Redirect to HTTPS
- **Allowed HTTP method** – `GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE`
- **Cache HTTP methods** – No
- **Allow gRPC requests over HTTP/2** – No
- **Restrict viewer access** – No
- **Cache policy** – `CachingDisabled` (Possible values: `UseOriginCacheControlHeaders`, `UseOriginCacheControlHeaders-QueryStrings`)
- **Origin request policy** – `AllViewerExceptHostHeader` (Possible values: `AllViewer`, `AllViewerandCloudFrontHeaders-2022-06`)
- **Response header policy** – None
- **Smooth Streaming** – No
- **Field level encryption** – No
- **Enable real-time access logs** – No
- **Functions** – No

Following are the settings that you can customize for your API Gateway origin in a multi-tenant distribution.

Customizable settings

- **Enable Origin Shield** – (Default: No)
- **Compress objects automatically** – (Default: Yes)

Custom origin and EC2 instance

Following are the origin settings that CloudFront preconfigures for your custom origin in a multi-tenant distribution.

Origin settings (preconfigured)

- **Protocol** – Match viewer

- **HTTP port** – 80
- **HTTPS port** – 443
- **Minimum origin SSL protocol** – TLSv1.2
- **Origin path** – None
- **Add custom header** – None
- **Enable Origin Shield** – No
- **Connection attempts** – 3
- **Response timeout** – 30
- **Keep-alive timeout** – 5

Following are the cache settings that CloudFront preconfigures for your custom origin and EC2 instance in a multi-tenant distribution.

Cache settings (preconfigured)

- **Compress objects automatically** – Yes
- **Viewer protocol policy** – Redirect to HTTPS
- **Allowed HTTP method** – `GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE`
- **Cache HTTP methods** – No
- **Allow gRPC requests over HTTP/2** – No
- **Restrict viewer access** – No
- **Cache policy** – `UseOriginCacheControlHeaders` (Possible values: `UseOriginCacheControlHeaders-QueryStrings`, `CachingDisabled`, `CachingOptimized`, `CachingOptimizedForUncompressedObjects`)
- **Origin request policy** – `AllViewer` (Possible values: `AllViewerExceptHostHeader`, `AllViewerandCloudFrontHeaders-2022-06`)
- **Response header policy** – None
- **Smooth Streaming** – No
- **Field level encryption** – No
- **Enable real-time access logs** – No
- **Functions** – No

Following are the settings that you can customize for your custom origin and EC2 instance in a multi-tenant distribution.

Customizable settings

- **Enable Origin Shield** – (Default: No)
- **Compress objects automatically** – (Default: Yes)
- **Caching** – (Default: `Cache by Default`)
  - If `Cache by Default` is selected, the `UseOriginCacheControlHeaders` cache policy is used.
  - If `Do Not Cache by Default` is selected, the `CachingDisabled` cache policy is used.
- **Include query string in cache** – (Default: Yes, if `Cache by Default` is already selected)
  - If `Do Not Cache by Default` is already selected and you then choose to include the query string in the cache, the `UseOriginCacheControlHeaders-QueryStrings` cache policy is used.

Elastic Load Balancing origin

Following are the origin settings that CloudFront preconfigures for your Elastic Load Balancing origin in a multi-tenant distribution.

Origin settings (preconfigured)

- **Protocol** – HTTPS only
- **HTTPS port** – 443
- **Minimum origin SSL protocol** – TLSv1.2
- **Origin path** – None
- **Add custom header** – None
- **Enable Origin Shield** – No
- **Connection attempts** – 3
- **Response timeout** – 30
- **Keep-alive timeout** – 5

Following are the cache settings that CloudFront preconfigures for your Elastic Load Balancing origin in a multi-tenant distribution.

Cache settings (preconfigured)

- **Compress objects automatically** – Yes
- **Viewer protocol policy** – Redirect to HTTPS
- **Allowed HTTP method** – `GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE`
- **Cache HTTP methods** – No
- **Allow gRPC requests over HTTP/2** – No
- **Restrict viewer access** – No
- **Caching** – (Default: `Cache by Default`)
  - If `Cache by Default` is selected, the `UseOriginCacheControlHeaders` cache policy is used.
  - If `Do Not Cache by Default` is selected, the `CachingDisabled` cache policy is used.
- **Include query string in cache** – (Default: Yes, if `Cache by Default` is already selected)
  - If `Do Not Cache by Default` is already selected and you then choose to include the query string in the cache, the `UseOriginCacheControlHeaders-QueryStrings` cache policy is used.
- **Origin request policy** – `AllViewer` (Possible values: `AllViewerExceptHostHeader`, `AllViewerandCloudFrontHeaders-2022-06`)
- **Response header policy** – None
- **Smooth Streaming** – No
- **Field level encryption** – No
- **Enable real-time access logs** – No
- **Functions** – No

Following are the settings that you can customize for your Elastic Load Balancing origin in a multi-tenant distribution.

Customizable settings

- **Enable Origin Shield** – (Default: No)
- **Compress objects automatically** – (Default: Yes)
- **Caching** – (Default: `Cache by Default`)

  - If `Cache by Default` is selected, the `UseOriginCacheControlHeaders` cache policy is used.
  - If `Do Not Cache by Default` is selected, the `CachingDisabled` cache policy is used.
- **Include query string in cache** – (Default: Yes, if `Cache by Default` is already selected)
  - If `Do Not Cache by Default` is already selected and you then choose to include the query string in the cache, the `UseOriginCacheControlHeaders-QueryStrings` cache policy is used.

MediaPackage v1 origin

Following are the origin settings that CloudFront preconfigures for your MediaPackage v1 origin in a multi-tenant distribution.

Origin settings (preconfigured)

- **Protocol** – HTTPS only
- **HTTPS port** – 443
- **Minimum origin SSL protocol** – TLSv1.2
- **Origin path** – You provide this by entering your MediaPackage URL.
- **Add custom header** – None
- **Enable Origin Shield** – No
- **Connection attempts** – 3
- **Response timeout** – 30
- **Keep-alive timeout** – 5

Following are the cache settings that CloudFront preconfigures for your MediaPackage v1 origin in a multi-tenant distribution.

Cache settings (preconfigured)

- **Compress objects automatically** – Yes
- **Viewer protocol policy** – Redirect to HTTPS
- **Allowed HTTP method** – `GET, HEAD`
- **Cache HTTP methods** – No

- **Allow gRPC requests over HTTP/2** – No
- **Restrict viewer access** – No
- **Cache policy** – `Elemental-MediaPackage`
- **Origin request policy** – None
- **Response header policy** – None
- **Smooth Streaming** – No
- **Field level encryption** – No
- **Enable real-time access logs** – No
- **Functions** – No

MediaPackage v2 origin

Following are the origin settings that CloudFront preconfigures for your MediaPackage v2 origin in a multi-tenant distribution.

Origin settings (preconfigured)

- **Origin Access Control** – CloudFront sets this up for you and adds the policy
- **Protocol** – HTTPS only
- **HTTPS port** – 443
- **Minimum origin SSL protocol** – TLSv1.2
- **Origin path** – None
- **Add custom header** – None
- **Enable Origin Shield** – No
- **Connection attempts** – 3
- **Response timeout** – 30
- **Keep-alive timeout** – 5

Following are the cache settings that CloudFront preconfigures for your MediaPackage v2 origin in a multi-tenant distribution.

Cache settings (preconfigured)

- **Compress objects automatically** – Yes

- **Viewer protocol policy** – Redirect to HTTPS
- **Allowed HTTP method** – `GET, HEAD`
- **Cache HTTP methods** – No
- **Allow gRPC requests over HTTP/2** – No
- **Restrict viewer access** – No
- **Cache policy** – `Elemental-MediaPackage`
- **Origin request policy** – None
- **Response header policy** – None
- **Smooth Streaming** – No
- **Field level encryption** – No
- **Enable real-time access logs** – No
- **Functions** – No

MediaTailor origin

Following are the origin settings that CloudFront preconfigures for your MediaTailor origin in a multi-tenant distribution.

Origin settings (preconfigured)

- **Protocol** – HTTPS only
- **HTTPS port** – 443
- **Minimum origin SSL protocol** – TLSv1.2
- **Origin path** – You provide this by entering your MediaPackage URL.
- **Add custom header** – None
- **Enable Origin Shield** – No
- **Connection attempts** – 3
- **Response timeout** – 30
- **Keep-alive timeout** – 5

Following are the cache settings that CloudFront preconfigures for your MediaTailor origin in a multi-tenant distribution.

Cache settings (preconfigured)

- **Compress objects automatically** – Yes
- **Viewer protocol policy** – Redirect to HTTPS
- **Allowed HTTP method** – `GET, HEAD`
- **Cache HTTP methods** – No
- **Allow gRPC requests over HTTP/2** – No
- **Restrict viewer access** – No
- **Cache policy** – None
- **Origin request policy** – `Elemental-MediaTailor-PersonalizedManifests`
- **Response header policy** – None
- **Smooth Streaming** – No
- **Field level encryption** – No
- **Enable real-time access logs** – No
- **Functions** – No

All distribution settings reference

You can choose to manually edit your CloudFront distribution settings when you create or update your distribution. Following are the settings that you can edit.

However, CloudFront configures most distribution settings for you, based on your content origin type. For more information, see Preconfigured distribution settings reference.

For more information about creating or updating a distribution by using the CloudFront console, see the section called “Create a distribution” or the section called “Update a distribution”.

Topics

- Origin settings
- Cache behavior settings
- Distribution settings
- Custom error pages and error caching
- Geographic restrictions

Origin settings

When you use the CloudFront console to create or update a distribution, you provide information about one or more locations, known as *origins*, where you store the original versions of your web content. CloudFront gets your web content from your origins and serves it to viewers by using a worldwide network of edge servers.

For the current maximum number of origins that you can create for a distribution, or to request a higher quota, see the section called “General quotas on distributions”.

If you want to delete an origin, you must first edit or delete the cache behaviors that are associated with that origin.

> **Important**
>
> If you delete an origin, confirm that files that were previously served by that origin are available in another origin and that your cache behaviors are now routing requests for those files to the new origin.

When you create or update a distribution, you specify the following values for each origin.

Topics

- Origin domain
- Protocol (custom origins only)
- Origin path
- Name
- Origin access (Amazon S3 origins only)
- Add custom header
- Enable Origin Shield
- Connection attempts
- Connection timeout
- Response timeout
- Response completion timeout
- Keep-alive timeout (custom and VPC origins only)
- Origin HTTP version (custom and VPC origins only)

- Response and keep-alive timeout quotas

Origin domain

The origin domain is the DNS domain name of the resource where CloudFront will get objects for your origin, such as an Amazon S3 bucket or HTTP server. For example:

- **Amazon S3 bucket** – *amzn-s3-demo-bucket*.s3.*us-west-2*.amazonaws.com

> **Note**
>
> If you recently created the S3 bucket, the CloudFront distribution might return HTTP `307 Temporary Redirect` responses for up to 24 hours. It can take up to 24 hours for the S3 bucket name to propagate to all AWS Regions. When the propagation is complete, the distribution automatically stops sending these redirect responses; you don't need to take any action. For more information, see Why am I getting an HTTP 307 Temporary Redirect response from Amazon S3? and Temporary Request Redirection.

- **Amazon S3 bucket configured as a website** – *amzn-s3-demo-bucket*.s3-website.*us-west-2*.amazonaws.com
- **MediaStore container** – *examplemediastore*.data.mediastore.*us-west-1*.amazonaws.com
- **MediaPackage endpoint** – *examplemediapackage*.mediapackage.*us-west-1*.amazonaws.com
- **Amazon EC2 instance** – *ec2-203-0-113-25*.compute-1.amazonaws.com
- **Elastic Load Balancing load balancer** – *example-load-balancer-1234567890*.*us-west-2*.elb.amazonaws.com
- **Your own web server** – www.example.com

Choose the domain name in the **Origin domain** field, or type the name. Resources from opt-in Regions must be entered manually. The domain name is not case-sensitive. Your origin domain must have a publicly resolvable DNS name that routes requests from clients to targets over the internet.

If you configure CloudFront to connect to your origin over HTTPS, one of the domain names in the certificate must match the domain name that you specify for **Origin Domain Name**. If no domain

name matches, CloudFront returns HTTP status code 502 (Bad Gateway) to the viewer. For more information, see Domain names in the CloudFront distribution and in the certificate and SSL/TLS negotiation failure between CloudFront and a custom origin server.

> **Note**
>
> If you are using an origin request policy that forwards the viewer host header to the origin, the origin must respond with a certificate that matches the viewer host header. For more information, see Add CloudFront request headers.

If your origin is an Amazon S3 bucket, note the following:

- If the bucket is configured as a website, enter the Amazon S3 static website hosting endpoint for your bucket; don't select the bucket name from the list in the **Origin domain** field. The static website hosting endpoint appears in the Amazon S3 console, on the **Properties** page under **Static website hosting**. For more information, see the section called "Use an Amazon S3 bucket that's configured as a website endpoint".
- If you configured Amazon S3 Transfer Acceleration for your bucket, do not specify the `s3-accelerate` endpoint for **Origin domain**.
- If you're using a bucket from a different AWS account and if the bucket is not configured as a website, enter the name, using the following format:

  bucket-name`.s3.`*region*`.amazonaws.com`

  If your bucket resides in a US Region, and you want Amazon S3 to route requests to a facility in northern Virginia, use the following format:

  bucket-name`.s3.us-east-1.amazonaws.com`
- The files must be publicly readable unless you secure your content in Amazon S3 by using a CloudFront origin access control. For more information about access control, see the section called "Restrict access to an Amazon S3 origin".

> **Important**
>
> If the origin is an Amazon S3 bucket, the bucket name must conform to DNS naming requirements. For more information, go to Bucket restrictions and limitations in the *Amazon Simple Storage Service User Guide*.

When you change the value of **Origin domain** for an origin, CloudFront immediately begins replicating the change to CloudFront edge locations. Until the distribution configuration is updated in a given edge location, CloudFront continues to forward requests to the previous origin. As soon as the distribution configuration is updated in that edge location, CloudFront begins to forward requests to the new origin.

Changing the origin does not require CloudFront to repopulate edge caches with objects from the new origin. As long as the viewer requests in your application have not changed, CloudFront continues to serve objects that are already in an edge cache until the TTL on each object expires or until seldom-requested objects are evicted.

Protocol (custom origins only)

> **Note**
>
> This applies only to custom origins.

The protocol policy that you want CloudFront to use when fetching objects from your origin.

Choose one of the following values:

- **HTTP only:** CloudFront uses only HTTP to access the origin.

  > **Important**
  >
  > **HTTP only** is the default setting when the origin is an Amazon S3 static website hosting endpoint, because Amazon S3 doesn't support HTTPS connections for static website hosting endpoints. The CloudFront console does not support changing this setting for Amazon S3 static website hosting endpoints.

- **HTTPS only:** CloudFront uses only HTTPS to access the origin.

- **Match viewer:** CloudFront communicates with your origin using HTTP or HTTPS, depending on the protocol of the viewer request. CloudFront caches the object only once even if viewers make requests using both HTTP and HTTPS protocols.

> **Important**
>
> For HTTPS viewer requests that CloudFront forwards to this origin, one of the domain names in the SSL/TLS certificate on your origin server must match the domain name that you specify for **Origin domain**. Otherwise, CloudFront responds to the viewer requests with an HTTP status code 502 (Bad Gateway) instead of returning the requested object. For more information, see the section called "Requirements for using SSL/TLS certificates with CloudFront".

Topics

- HTTP port
- HTTPS port
- Minimum origin SSL protocol

HTTP port

> **Note**
>
> This applies only to custom origins.

(Optional) You can specify the HTTP port on which the custom origin listens. Valid values include ports 80, 443, and 1024 to 65535. The default value is port 80.

> **Important**
>
> Port 80 is the default setting when the origin is an Amazon S3 static website hosting endpoint, because Amazon S3 only supports port 80 for static website hosting endpoints. The CloudFront console does not support changing this setting for Amazon S3 static website hosting endpoints.

HTTPS port

> **Note**
>
> This applies only to custom origins.

(Optional) You can specify the HTTPS port on which the custom origin listens. Valid values include ports 80, 443, and 1024 to 65535. The default value is port 443. When **Protocol** is set to **HTTP only**, you cannot specify a value for **HTTPS port**.

Minimum origin SSL protocol

> **Note**
>
> This applies only to custom origins.

Choose the minimum TLS/SSL protocol that CloudFront can use when it establishes an HTTPS connection to your origin. Lower TLS protocols are less secure, so we recommend that you choose the latest TLS protocol that your origin supports. When **Protocol** is set to **HTTP only**, you cannot specify a value for **Minimum origin SSL protocol**.

If you use the CloudFront API to set the TLS/SSL protocol for CloudFront to use, you cannot set a minimum protocol. Instead, you specify all of the TLS/SSL protocols that CloudFront can use with your origin. For more information, see OriginSslProtocols in the *Amazon CloudFront API Reference*.

Origin path

If you want CloudFront to request your content from a directory in your origin, enter the directory path, beginning with a slash (/). CloudFront appends the directory path to the value of **Origin domain**, for example, **`cf-origin.example.com/production/images`**. Do not add a slash (/) at the end of the path.

For example, suppose you've specified the following values for your distribution:

- **Origin domain** – An Amazon S3 bucket named **`amzn-s3-demo-bucket`**
- **Origin path** – **`/production`**

- **Alternate domain names (CNAME) – `example.com`**

When a user enters `example.com/index.html` in a browser, CloudFront sends a request to Amazon S3 for `amzn-s3-demo-bucket/production/index.html`.

When a user enters `example.com/acme/index.html` in a browser, CloudFront sends a request to Amazon S3 for `amzn-s3-demo-bucket/production/acme/index.html`.

Name

A name is a string that uniquely identifies this origin in this distribution. You might create cache behaviors in addition to the default cache behavior. In this case, you use the name that you specify here to identify the origin that you want CloudFront to route a request to when the request matches the path pattern for that cache behavior.

Origin access (Amazon S3 origins only)

> **Note**
>
> This applies only to Amazon S3 bucket origins (those that are *not* using the S3 static website endpoint).

Choose **Origin access control settings (recommended)** if you want to make it possible to restrict access to an Amazon S3 bucket origin to only specific CloudFront distributions.

Choose **Public** if the Amazon S3 bucket origin is publicly accessible.

For more information, see the section called “Restrict access to an Amazon S3 origin”.

For information about how to require users to access objects on a custom origin by using only CloudFront URLs, see the section called “Restrict access to files on custom origins”.

Add custom header

If you want CloudFront to add custom headers whenever it sends a request to your origin, specify the header name and its value. For more information, see the section called “Add custom headers to origin requests”.

For the current maximum number of custom headers that you can add, the maximum length of a custom header name and value, and the maximum total length of all header names and values, see *Quotas*.

Enable Origin Shield

Choose **Yes** to enable CloudFront Origin Shield. For more information about Origin Shield, see the section called "Use Origin Shield".

Connection attempts

You can set the number of times that CloudFront attempts to connect to the origin. You can specify 1, 2, or 3 as the number of attempts. The default number (if you don't specify otherwise) is 3.

Use this setting together with **Connection timeout** to specify how long CloudFront waits before attempting to connect to the secondary origin or returning an error response to the viewer. By default, CloudFront waits as long as 30 seconds (3 attempts of 10 seconds each) before attempting to connect to the secondary origin or returning an error response. You can reduce this time by specifying fewer attempts, a shorter connection timeout, or both.

If the specified number of connection attempts fail, CloudFront does one of the following:

- If the origin is part of an origin group, CloudFront attempts to connect to the secondary origin. If the specified number of connection attempts to the secondary origin fail, then CloudFront returns an error response to the viewer.
- If the origin is not part of an origin group, CloudFront returns an error response to the viewer.

For a custom origin (including an Amazon S3 bucket that's configured with static website hosting), this setting also specifies the number of times that CloudFront attempts to get a response from the origin. For more information, see the section called "Response timeout".

Connection timeout

The connection timeout is the number of seconds that CloudFront waits when trying to establish a connection to the origin. You can specify a number of seconds between 1 and 10 (inclusive). The default timeout (if you don't specify otherwise) is 10 seconds.

Use this setting together with **Connection attempts** to specify how long CloudFront waits before attempting to connect to the secondary origin or before returning an error response to the viewer. By default, CloudFront waits as long as 30 seconds (3 attempts of 10 seconds each) before

attempting to connect to the secondary origin or returning an error response. You can reduce this time by specifying fewer attempts, a shorter connection timeout, or both.

If CloudFront doesn't establish a connection to the origin within the specified number of seconds, CloudFront does one of the following:

- If the specified number of **Connection attempts** is more than 1, CloudFront tries again to establish a connection. CloudFront tries up to 3 times, as determined by the value of **Connection attempts**.
- If all the connection attempts fail and the origin is part of an origin group, CloudFront attempts to connect to the secondary origin. If the specified number of connection attempts to the secondary origin fail, then CloudFront returns an error response to the viewer.
- If all the connection attempts fail and the origin is not part of an origin group, CloudFront returns an error response to the viewer.

Response timeout

The origin response timeout, also known as the *origin read timeout* or *origin request timeout*, applies to both of the following values:

- How long (in seconds) CloudFront waits for a response after forwarding a request to the origin.
- How long (in seconds) CloudFront waits after receiving a packet of a response from the origin and before receiving the next packet.

> **Tip**
>
> If you want to increase the timeout value because viewers are experiencing HTTP 504 status code errors, consider exploring other ways to eliminate those errors before changing the timeout value. See the troubleshooting suggestions in the section called "HTTP 504 status code (Gateway Timeout)".

CloudFront behavior depends on the HTTP method in the viewer request:

- `GET` and `HEAD` requests – If the origin doesn't respond or stops responding within the duration of the response timeout, CloudFront drops the connection. CloudFront tries again to connect according to the value of the section called "Connection attempts".

- `DELETE`, `OPTIONS`, `PATCH`, `PUT`, and `POST` requests – If the origin doesn't respond for the duration of the read timeout, CloudFront drops the connection and doesn't try again to contact the origin. The client can resubmit the request if necessary.

Response completion timeout

> **Note**
>
> Response completion timeout doesn't support the continuous deployment feature.

The time (in seconds) that a request from CloudFront to the origin can stay open and wait for a response. If the complete response isn't received from the origin by this time, CloudFront ends the connection.

Unlike **Response timeout**, which is the wait time for *individual* response packets, **Response completion timeout** is the *maximum* allowed amount of time that CloudFront waits for the response to complete. You can use this setting to make sure that CloudFront doesn't wait indefinitely for a slow or unresponsive origin, even if other timeout settings allow for a longer wait.

This maximum timeout includes what you specified for other timeout settings and the number of **Connection attempts** for each retry. You can use these settings together to specify how long CloudFront waits for the full request and when to end the request, regardless if it's complete or not.

For example, if you set the following settings:

- **Connection attempts** is 3
- **Connection timeout** is 10 seconds
- **Response timeout** is 30 seconds
- **Response completion timeout** is 60 seconds

This means CloudFront will try to connect to the origin (up to 3 total attempts), with each connection attempt timing out at 10 seconds. Once connected, CloudFront will wait up to 30 seconds for the origin to respond to the request until it receives the last packet of the response.

Regardless of the number of connection attempts or the response timeout, CloudFront will end the connection if the complete response from the origin takes longer than 60 seconds to complete.

CloudFront will then return to the viewer an the section called "HTTP 504 status code (Gateway Timeout)" error response or a custom error response if you specified one.

Notes

- If you set a value for response completion timeout, the value must be equal to or greater than the value for the response timeout (origin read timeout).
- If you don't set a value for the response completion timeout, CloudFront doesn't enforce a maximum value.

Keep-alive timeout (custom and VPC origins only)

The keep-alive timeout is how long (in seconds) CloudFront tries to maintain a connection to your custom origin after it gets the last packet of a response. Maintaining a persistent connection saves the time that is required to re-establish the TCP connection and perform another TLS handshake for subsequent requests. Increasing the keep-alive timeout helps improve the request-per-connection metric for distributions.

Note

For the **Keep-alive timeout** value to have an effect, your origin must be configured to allow persistent connections.

Origin HTTP version (custom and VPC origins only)

The HTTP version that CloudFront uses when it connects to your custom origin or VPC origin. Choose one of the following values:

- **HTTP/1.1** – CloudFront uses HTTP/1.1 for all connections to the origin. This is the default.
- **HTTP/2** – CloudFront offers HTTP/2 to the origin when it establishes the TLS connection. If the origin accepts HTTP/2, CloudFront can send multiple requests over the same connection, which reduces the number of connections and TLS handshakes that your origin handles. If the origin doesn't support HTTP/2, CloudFront uses HTTP/1.1, so you don't need to change the origin before you choose this value.

CloudFront uses HTTP/2 only for HTTPS connections, and only when the origin's minimum SSL protocol is TLSv1.2 or later. Even when you choose **HTTP/2**, CloudFront uses HTTP/1.1 in the following cases:

- **Protocol** is **HTTP only**.
- **Protocol** is **Match viewer**, and the viewer request uses HTTP.
- **Minimum origin SSL protocol** is earlier than TLSv1.2 (in the API, `OriginSslProtocols` includes `SSLv3`, `TLSv1`, or `TLSv1.1`). This applies to every connection to the origin, even connections that use TLS 1.2 or later. To use HTTP/2, set the minimum to TLSv1.2. For more information, see Minimum origin SSL protocol.

If a cache behavior has a Lambda@Edge function for the origin request or origin response event, CloudFront might use HTTP/1.1 for requests that match that cache behavior.

For VPC origins, you specify the HTTP version for the origin in each distribution, not on the VPC origin resource. Distributions that use the same VPC origin can use different HTTP versions.

In the CloudFront API, this setting is the `OriginHttpVersion` field in CustomOriginConfig and VpcOriginConfig. Valid values are `http1.1` and `http2`. If you update a distribution and omit `OriginHttpVersion` for an origin, CloudFront uses HTTP/1.1 for that origin.

Response and keep-alive timeout quotas

- For response timeout, the default is 30 seconds.
- For keep-alive timeout, the default is 5 seconds.

If you request a timeout increase for your AWS account, update your distribution origins so that they have the response timeout and keep-alive timeout values that you want. A quota increase for your account doesn't automatically update your origins. For example, if you're using a Lambda@Edge function to set a keep-alive timeout of 90 seconds, your origin must already have a keep-alive timeout of 90 seconds or more. Otherwise, your Lambda@Edge function might fail to execute.

For more information about distribution quotas, including how to request an increase, see General quotas on distributions.

Cache behavior settings

By setting the cache behavior, you can configure a variety of CloudFront functionality for a given URL path pattern for files on your website. For example, one cache behavior might apply to all `.jpg` files in the `images` directory on a web server that you're using as an origin server for CloudFront. The functionality that you can configure for each cache behavior includes:

- The path pattern
- If you have configured multiple origins for your CloudFront distribution, the origin to which you want CloudFront to forward your requests
- Whether to forward query strings to your origin
- Whether accessing the specified files requires signed URLs
- Whether to require users to use HTTPS to access those files
- The minimum amount of time that those files stay in the CloudFront cache regardless of the value of any `Cache-Control` headers that your origin adds to the files

When you create a new distribution, you specify settings for the default cache behavior, which automatically forwards all requests to the origin that you specify when you create the distribution. After you create a distribution, you can create additional cache behaviors that define how CloudFront responds when it receives a request for objects that match a path pattern, for example, `*.jpg`. If you create additional cache behaviors, the default cache behavior is always the last to be processed. Other cache behaviors are processed in the order in which they're listed in the CloudFront console or, if you're using the CloudFront API, the order in which they're listed in the `DistributionConfig` element for the distribution. For more information, see Path pattern.

When you create a cache behavior, you specify the one origin from which you want CloudFront to get objects. As a result, if you want CloudFront to distribute objects from all of your origins, you must have at least as many cache behaviors (including the default cache behavior) as you have origins. For example, if you have two origins and only the default cache behavior, the default cache behavior causes CloudFront to get objects from one of the origins, but the other origin is never used.

For the current maximum number of cache behaviors that you can add to a distribution, or to request a higher quota (formerly known as limit), see General quotas on distributions.

Topics

- Path pattern

- Origin or origin group
- Viewer protocol policy
- Allowed HTTP methods
- Field-level encryption config
- Cached HTTP methods
- Allow gRPC requests over HTTP/2
- Cache based on selected request headers
- Allowlist headers
- Object caching
- Minimum TTL
- Maximum TTL
- Default TTL
- Forward cookies
- Allowlist cookies
- Query string forwarding and caching
- Query string allowlist
- Smooth Streaming
- Restrict viewer access (use signed URLs or signed cookies)
- Trusted signers
- AWS account numbers
- Compress objects automatically
- CloudFront event
- Lambda function ARN
- Include body

Path pattern

A path pattern (for example, `images/*.jpg`) specifies to which requests you want this cache behavior to apply. When CloudFront receives an end-user request, the requested path is compared with path patterns in the order in which cache behaviors are listed in the distribution. The first

match determines which cache behavior is applied to that request. For example, suppose you have three cache behaviors with the following three path patterns, in this order:

- `images/*.jpg`
- `images/*`
- `*.gif`

> **Note**
>
> You can optionally include a slash (/) at the beginning of the path pattern, for example, `/images/*.jpg`. CloudFront behavior is the same with or without the leading /. If you don't specify the / at the beginning of the path, this character is automatically implied; CloudFront treats the path the same with or without the leading /. For example, CloudFront treats `/*product.jpg` the same as `*product.jpg`

A request for the file `images/sample.gif` doesn't satisfy the first path pattern, so the associated cache behaviors are not applied to the request. The file does satisfy the second path pattern, so the cache behaviors associated with the second path pattern are applied even though the request also matches the third path pattern.

> **Note**
>
> When you create a new distribution, the value of **Path Pattern** for the default cache behavior is set to * (all files) and cannot be changed. This value causes CloudFront to forward all requests for your objects to the origin that you specified in the Origin domain field. If the request for an object does not match the path pattern for any of the other cache behaviors, CloudFront applies the behavior that you specify in the default cache behavior.

> **Important**
>
> Define path patterns and their sequence carefully or you may give users undesired access to your content. For example, suppose a request matches the path pattern for two cache behaviors. The first cache behavior does not require signed URLs and the second cache behavior does require signed URLs. Users are able to access the objects without using a

signed URL because CloudFront processes the cache behavior associated with the first match.

If you're working with a MediaPackage channel, you must include specific path patterns for the cache behavior that you define for the endpoint type for your origin. For example, for a DASH endpoint, you type `*.mpd` for **Path Pattern**. For more information and specific instructions, see Serve live video formatted with AWS Elemental MediaPackage.

The path you specify applies to requests for all files in the specified directory and in subdirectories below the specified directory. CloudFront does not consider query strings or cookies when evaluating the path pattern. For example, if an `images` directory contains `product1` and `product2` subdirectories, the path pattern `images/*.jpg` applies to requests for any .jpg file in the `images`, `images/product1`, and `images/product2` directories. If you want to apply a different cache behavior to the files in the `images/product1` directory than the files in the `images` and `images/product2` directories, create a separate cache behavior for `images/product1` and move that cache behavior to a position above (before) the cache behavior for the `images` directory.

You can use the following wildcard characters in your path pattern:

- `*` matches 0 or more characters.
- `?` matches exactly 1 character.

The following examples show how the wildcard characters work:

Path pattern	Files that match the path pattern
`*.jpg`	All .jpg files.
`images/*.jpg`	All .jpg files in the `images` directory and in subdirectories under the `images` directory.
`a*.jpg`	• All .jpg files for which the file name begins with a, for example, `apple.jpg` and `appalachian_trail_2012_05_21.jpg` .

Path pattern	Files that match the path pattern
	• All .jpg files for which the file path begins with a, for example, abra/cadabra/magic.jpg .
a??.jpg	All .jpg files for which the file name begins with a and is followed by exactly two other characters, for example, ant.jpg and abe.jpg.
.doc	All files for which the file name extension begins with .doc, for example, .doc, .docx, and .docm files. You can't use the path pattern *.doc? in this case, because that path pattern wouldn't apply to requests for .doc files; the ? wildcard character replaces exactly one character.

The maximum length of a path pattern is 255 characters. The value can contain any of the following characters:

- A-Z, a-z

  Path patterns are case-sensitive, so the path pattern *.jpg doesn't apply to the file LOGO.JPG
- 0-9
- _ - . * $ / ~ " ' @ : +
- &, passed and returned as &

Path normalization

CloudFront normalizes URI paths consistent with RFC 3986 and then matches the path with the correct cache behavior. After the cache behavior is matched, CloudFront sends the raw URI path to the origin. If they don't match, requests are instead matched to your default cache behavior.

Some characters are normalized and removed from the path, such as multiple slashes (//) or periods (..). This can alter the URL that CloudFront uses to match the intended cache behavior.

Example

You specify the /a/b* and /a* paths for your cache behavior.

- A viewer sending the `/a/b?c=1` path will match the `/a/b*` cache behavior.
- A viewer sending the `/a/b/..?c=1` path will match the `/a*` cache behavior.

To work around the paths being normalized, you can update your request paths or the path pattern for the cache behavior.

Origin or origin group

This setting applies only when you create or update a cache behavior for an existing distribution.

Enter the value of an existing origin or origin group. This identifies the origin or origin group to which you want CloudFront to route requests when a request (such as https://example.com/logo.jpg) matches the path pattern for a cache behavior (such as *.jpg) or for the default cache behavior (*).

Viewer protocol policy

Choose the protocol policy that you want viewers to use to access your content in CloudFront edge locations:

- **HTTP and HTTPS**: Viewers can use both protocols.
- **Redirect HTTP to HTTPS**: Viewers can use both protocols, but HTTP requests are automatically redirected to HTTPS requests.
- **HTTPS Only**: Viewers can only access your content if they're using HTTPS.

For more information, see Require HTTPS for communication between viewers and CloudFront.

Allowed HTTP methods

Specify the HTTP methods that you want CloudFront to process and forward to your origin:

- **GET, HEAD:** You can use CloudFront only to get objects from your origin or to get object headers.
- **GET, HEAD, OPTIONS:** You can use CloudFront only to get objects from your origin, get object headers, or retrieve a list of the options that your origin server supports.
- **GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE:** You can use CloudFront to get, add, update, and delete objects, and to get object headers. In addition, you can perform other POST operations such as submitting data from a web form.

Note

If you're using gRPC in your workload, you must select **GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE**. gRPC workloads require the `POST` method. For more information, see Using gRPC with CloudFront distributions.
CloudFront caches responses to `GET` and `HEAD` requests and, optionally, `OPTIONS` requests. Responses to `OPTIONS` requests are cached separately from responses to `GET` and `HEAD` requests (the `OPTIONS` method is included in the cache key for `OPTIONS` requests). CloudFront does not cache responses to requests that use other methods.

Important

If you choose **GET, HEAD, OPTIONS** or **GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE**, you might need to restrict access to your Amazon S3 bucket or to your custom origin to prevent users from performing operations that you don't want them to perform. The following examples explain how to restrict access:

- **If you're using Amazon S3 as an origin for your distribution:** Create a CloudFront origin access control to restrict access to your Amazon S3 content, and give permissions to the origin access control. For example, if you configure CloudFront to accept and forward these methods *only* because you want to use `PUT`, you must still configure Amazon S3 bucket policies to handle `DELETE` requests appropriately. For more information, see Restrict access to an Amazon S3 origin.
- **If you're using a custom origin:** Configure your origin server to handle all methods. For example, if you configure CloudFront to accept and forward these methods *only* because you want to use `POST`, you must still configure your origin server to handle `DELETE` requests appropriately.

Field-level encryption config

If you want to enforce field-level encryption on specific data fields, in the dropdown list, choose a field-level encryption configuration.

For more information, see Use field-level encryption to help protect sensitive data.

Cached HTTP methods

Specify whether you want CloudFront to cache the response from your origin when a viewer submits an OPTIONS request. CloudFront always caches the response to GET and HEAD requests.

Allow gRPC requests over HTTP/2

Specify whether you want your distribution to allow gRPC requests. To enable gRPC, select the following settings:

- For **Allowed HTTP methods**, select the **GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE** methods. gRPC requires the POST method.
- Select the gRPC checkbox that appears after you select the POST method.
- For **Supported HTTP versions**, select **HTTP/2**.

For more information, see Using gRPC with CloudFront distributions.

Cache based on selected request headers

Specify whether you want CloudFront to cache objects based on the values of specified headers:

- **None (improves caching)** – CloudFront doesn't cache your objects based on header values.
- **Allowlist** – CloudFront caches your objects based only on the values of the specified headers. Use **Allowlist Headers** to choose the headers that you want CloudFront to base caching on.
- **All** – CloudFront doesn't cache the objects that are associated with this cache behavior. Instead, CloudFront sends every request to the origin. (Not recommended for Amazon S3 origins.)

Regardless of the option that you choose, CloudFront forwards certain headers to your origin and takes specific actions based on the headers that you forward. For more information about how CloudFront handles header forwarding, see HTTP request headers and CloudFront behavior (custom and Amazon S3 origins).

For more information about how to configure caching in CloudFront by using request headers, see Cache content based on request headers.

Allowlist headers

These settings apply only when you choose **Allowlist** for **Cache Based on Selected Request Headers**.

Specify the headers that you want CloudFront to consider when caching your objects. Select headers from the list of available headers and choose **Add**. To forward a custom header, enter the name of the header in the field, and choose **Add Custom**.

For the current maximum number of headers that you can allowlist for each cache behavior, or to request a higher quota (formerly known as limit), see Quotas on headers.

Object caching

If your origin server is adding a `Cache-Control` header to your objects to control how long the objects stay in the CloudFront cache and if you don't want to change the `Cache-Control` value, choose **Use Origin Cache Headers**.

To specify a minimum and maximum time that your objects stay in the CloudFront cache regardless of `Cache-Control` headers, and a default time that your objects stay in the CloudFront cache when the `Cache-Control` header is missing from an object, choose **Customize**. Then specify values in the **Minimum TTL**, **Default TTL**, and **Maximum TTL** fields.

For more information, see Manage how long content stays in the cache (expiration).

Minimum TTL

Specify the minimum amount of time, in seconds, that you want objects to stay in the CloudFront cache before CloudFront sends another request to the origin to determine whether the object has been updated.

> **Warning**
>
> If your minimum TTL is greater than 0, CloudFront will cache content for at least the duration specified in the cache policy's minimum TTL, even if the `Cache-Control: no-cache`, `no-store`, or `private` directives are present in the origin headers.

For more information, see Manage how long content stays in the cache (expiration).

Maximum TTL

Specify the maximum amount of time, in seconds, that you want objects to stay in CloudFront caches before CloudFront queries your origin to see whether the object has been updated. The

value that you specify for **Maximum TTL** applies only when your origin adds HTTP headers such as `Cache-Control max-age`, `Cache-Control s-maxage`, or `Expires` to objects. For more information, see Manage how long content stays in the cache (expiration).

To specify a value for **Maximum TTL**, you must choose the **Customize** option for the **Object Caching** setting.

The default value for **Maximum TTL** is 31536000 seconds (one year). If you change the value of **Minimum TTL** or **Default TTL** to more than 31536000 seconds, then the default value of **Maximum TTL** changes to the value of **Default TTL**.

Default TTL

Specify the default amount of time, in seconds, that you want objects to stay in CloudFront caches before CloudFront forwards another request to your origin to determine whether the object has been updated. The value that you specify for **Default TTL** applies only when your origin does *not* add HTTP headers such as `Cache-Control max-age`, `Cache-Control s-maxage`, or `Expires` to objects. For more information, see Manage how long content stays in the cache (expiration).

To specify a value for **Default TTL**, you must choose the **Customize** option for the **Object Caching** setting.

The default value for **Default TTL** is 86400 seconds (one day). If you change the value of **Minimum TTL** to more than 86400 seconds, then the default value of **Default TTL** changes to the value of **Minimum TTL**.

Forward cookies

> **Note**
>
> For Amazon S3 origins, this option applies to only buckets that are configured as a website endpoint.

Specify whether you want CloudFront to forward cookies to your origin server and, if so, which ones. If you choose to forward only selected cookies (an allowlist of cookies), enter the cookie names in the **Allowlist Cookies** field. If you choose **All**, CloudFront forwards all cookies regardless of how many your application uses.

Amazon S3 doesn't process cookies, and forwarding cookies to the origin reduces cache ability. For cache behaviors that are forwarding requests to an Amazon S3 origin, choose **None** for **Forward Cookies**.

For more information about forwarding cookies to the origin, go to Cache content based on cookies.

Allowlist cookies

> **Note**
>
> For Amazon S3 origins, this option applies to only buckets that are configured as a website endpoint.

If you chose **Allowlist** in the **Forward Cookies** list, then in the **Allowlist Cookies** field, enter the names of cookies that you want CloudFront to forward to your origin server for this cache behavior. Enter each cookie name on a new line.

You can specify the following wildcards to specify cookie names:

- ***** matches 0 or more characters in the cookie name
- **?** matches exactly one character in the cookie name

For example, suppose viewer requests for an object include a cookie named:

`userid_`*`member-number`*

Where each of your users has a unique value for *`member-number`*. You want CloudFront to cache a separate version of the object for each member. You could accomplish this by forwarding all cookies to your origin, but viewer requests include some cookies that you don't want CloudFront to cache. Alternatively, you could specify the following value as a cookie name, which causes CloudFront to forward to the origin all of the cookies that begin with `userid_`:

```
userid_*
```

For the current maximum number of cookie names that you can allowlist for each cache behavior, or to request a higher quota (formerly known as limit), see Quotas on cookies (legacy cache settings).

Query string forwarding and caching

CloudFront can cache different versions of your content based on the values of query string parameters. Choose one of the following options:

None (Improves Caching)

Choose this option if your origin returns the same version of an object regardless of the values of query string parameters. This increases the likelihood that CloudFront can serve a request from the cache, which improves performance and reduces the load on your origin.

Forward all, cache based on allowlist

Choose this option if your origin server returns different versions of your objects based on one or more query string parameters. Then specify the parameters that you want CloudFront to use as a basis for caching in the Query string allowlist field.

Forward all, cache based on all

Choose this option if your origin server returns different versions of your objects for all query string parameters.

For more information about caching based on query string parameters, including how to improve performance, see Cache content based on query string parameters.

Query string allowlist

This setting applies only when you choose **Forward all, cache based on allowlist** for Query string forwarding and caching. You can specify the query string parameters that you want CloudFront to use as a basis for caching.

Smooth Streaming

Choose **Yes** if you want to distribute media files in the Microsoft Smooth Streaming format and you do not have an IIS server.

Choose **No** if you have a Microsoft IIS server that you want to use as an origin to distribute media files in the Microsoft Smooth Streaming format, or if you are not distributing Smooth Streaming media files.

> **Note**
>
> If you specify **Yes**, you can still distribute other content using this cache behavior if that content matches the value of **Path Pattern**.

For more information, see Configure video on demand for Microsoft Smooth Streaming.

Restrict viewer access (use signed URLs or signed cookies)

If you want requests for objects that match the `PathPattern` for this cache behavior to use public URLs, choose **No**.

If you want requests for objects that match the `PathPattern` for this cache behavior to use signed URLs, choose **Yes**. Then specify the AWS accounts that you want to use to create signed URLs; these accounts are known as trusted signers.

For more information about trusted signers, see Specify signers that can create signed URLs and signed cookies.

Trusted signers

This setting applies only when you choose **Yes** for **Restrict Viewer Access (Use Signed URLs or Signed Cookies**.

Choose which AWS accounts you want to use as trusted signers for this cache behavior:

- **Self:** Use the account with which you're currently signed into the AWS Management Console as a trusted signer. If you're currently signed in as an IAM user, the associated AWS account is added as a trusted signer.
- **Specify Accounts:** Enter account numbers for trusted signers in the **AWS Account Numbers** field.

To create signed URLs, an AWS account must have at least one active CloudFront key pair.

> **Important**
>
> If you're updating a distribution that you're already using to distribute content, add trusted signers only when you're ready to start generating signed URLs for your objects. After you

add trusted signers to a distribution, users must use signed URLs to access the objects that match the `PathPattern` for this cache behavior.

AWS account numbers

This setting applies only when you choose **Specify Accounts** for **Trusted Signers**.

If you want to create signed URLs using AWS accounts in addition to or instead of the current account, enter one AWS account number per line in this field. Note the following:

- The accounts that you specify must have at least one active CloudFront key pair. For more information, see Create key pairs for your signers.
- You can't create CloudFront key pairs for IAM users, so you can't use IAM users as trusted signers.
- For information about how to get the AWS account number for an account, see View AWS account identifiers in the *AWS Account Management Reference Guide*.
- If you enter the account number for the current account, CloudFront automatically checks the **Self** check box and removes the account number from the **AWS Account Numbers** list.

Compress objects automatically

If you want CloudFront to automatically compress files of certain types when viewers support compressed content, choose **Yes**. When CloudFront compresses your content, downloads are faster because the files are smaller, and your web pages render faster for your users. For more information, see Serve compressed files.

CloudFront event

This setting applies to **Lambda Function Associations**.

You can choose to run a Lambda function when one or more of the following CloudFront events occur:

- When CloudFront receives a request from a viewer (viewer request)
- Before CloudFront forwards a request to the origin (origin request)
- When CloudFront receives a response from the origin (origin response)
- Before CloudFront returns the response to the viewer (viewer response)

For more information, see Choose the event to trigger the function.

Lambda function ARN

This setting applies to **Lambda Function Associations**.

Specify the Amazon Resource Name (ARN) of the Lambda function that you want to add a trigger for. To learn how to get the ARN for a function, see step 1 of the procedure Adding Triggers by Using the CloudFront Console.

Include body

This setting applies to **Lambda Function Associations**.

For more information, see Include body.

Distribution settings

The following values apply to the entire distribution.

Topics

- Price class
- AWS WAF web ACL
- Alternate domain names (CNAMEs)
- SSL certificate
- Custom SSL client support
- Security policy (minimum SSL/TLS version)
- Supported HTTP versions
- Default root object
- Standard logging
- Connection logs
- Log prefix
- Cookie logging
- Enable IPv6 (viewer requests)
- Mutual authentication
- Enable IPv6 for custom origins (origin requests)
- Comment

- Distribution state

Price class

Choose the price class that corresponds with the maximum price that you want to pay for CloudFront service. By default, CloudFront serves your objects from edge locations in all CloudFront Regions.

For more information about price classes and about how your choice of price class affects CloudFront performance for your distribution, see CloudFront pricing.

AWS WAF web ACL

You can protect your CloudFront distribution with AWS WAF, a web application firewall that allows you to secure your web applications and APIs to block requests before they reach your servers. You can Enable AWS WAF for distributions when creating or editing a CloudFront distribution.

Optionally, you can later configure additional security protections for other threats specific to your application in the AWS WAF console at https://eusc-de-east-1.console.amazonaws-eusc.eu/wafv2/.

For more information about AWS WAF, see the AWS WAF Developer Guide.

Alternate domain names (CNAMEs)

Optional. Specify one or more domain names that you want to use for URLs for your objects instead of the domain name that CloudFront assigns when you create your distribution. You must own the domain name, or have authorization to use it, which you verify by adding an SSL/TLS certificate.

For example, if you want the URL for the object:

`/images/image.jpg`

To look like this:

`https://www.example.com/images/image.jpg`

Instead of like this:

`https://d111111abcdef8.cloudfront.net/images/image.jpg`

Add a CNAME for `www.example.com`.

Important

If you add a CNAME for `www.example.com` to your distribution, you also must do the following:

- Create (or update) a CNAME record with your DNS service to route queries for `www.example.com` to `d111111abcdef8.cloudfront.net`.
- Add a certificate to CloudFront from a trusted certificate authority (CA) that covers the domain name (CNAME) that you add to your distribution, to validate your authorization to use the domain name.

You must have permission to create a CNAME record with the DNS service provider for the domain. Typically, this means that you own the domain, or that you're developing an application for the domain owner.

For the current maximum number of alternate domain names that you can add to a distribution, or to request a higher quota (formerly known as limit), see General quotas on distributions.

For more information about alternate domain names, see Use custom URLs by adding alternate domain names (CNAMEs). For more information about CloudFront URLs, see Customize the URL format for files in CloudFront.

SSL certificate

If you specified an alternate domain name to use with your distribution, choose **Custom SSL Certificate**, and then, to validate your authorization to use the alternate domain name, choose a certificate that covers it. If you want viewers to use HTTPS to access your objects, choose the settings that support that.

- **Default CloudFront Certificate (*.cloudfront.net)** – Choose this option if you want to use the CloudFront domain name in the URLs for your objects, such as `https://d111111abcdef8.cloudfront.net/image1.jpg`.
- **Custom SSL Certificate** – Choose this option if you want to use your own domain name in the URLs for your objects as an alternate domain name, such as `https://example.com/image1.jpg`. Then choose a certificate to use that covers the alternate domain name. The list of certificates can include any of the following:

- Certificates provided by AWS Certificate Manager
- Certificates that you purchased from a third-party certificate authority and uploaded to ACM
- Certificates that you purchased from a third-party certificate authority and uploaded to the IAM certificate store

If you choose this setting, we recommend that you use only an alternate domain name in your object URLs (https://example.com/logo.jpg). If you use your CloudFront distribution domain name (https://d111111abcdef8.cloudfront.net/logo.jpg) and a client uses an older viewer that doesn't support SNI, how the viewer responds depends on the value that you choose for **Clients Supported**:

- **All Clients**: The viewer displays a warning because the CloudFront domain name doesn't match the domain name in your SSL/TLS certificate.
- **Only Clients that Support Server Name Indication (SNI)**: CloudFront drops the connection with the viewer without returning the object.

Custom SSL client support

Applies only when you choose **Custom SSL Certificate (example.com)** for **SSL Certificate**. If you specified one or more alternate domain names and a custom SSL certificate for the distribution, choose how you want CloudFront to serve HTTPS requests:

- **Clients that Support Server Name Indication (SNI) - (Recommended)** – With this setting, virtually all modern web browsers and clients can connect to the distribution, because they support SNI. However, some viewers might use older web browsers or clients that don't support SNI, which means they can't connect to the distribution.

  To apply this setting using the CloudFront API, specify `sni-only` in the `SSLSupportMethod` field. In CloudFormation, the field is named `SslSupportMethod` (note the different capitalization).
- **Legacy Clients Support** – With this setting, older web browsers and clients that don't support SNI can connect to the distribution. However, this setting incurs additional monthly charges. For the exact price, go to the Amazon CloudFront Pricing page, and search the page for **Dedicated IP custom SSL**.

  To apply this setting using the CloudFront API, specify `vip` in the `SSLSupportMethod` field. In CloudFormation, the field is named `SslSupportMethod` (note the different capitalization).

For more information, see Choose how CloudFront serves HTTPS requests.

Security policy (minimum SSL/TLS version)

Specify the security policy that you want CloudFront to use for HTTPS connections with viewers (clients). A security policy determines two settings:

- The minimum SSL/TLS protocol that CloudFront uses to communicate with viewers.
- The ciphers that CloudFront can use to encrypt the content that it returns to viewers.

For more information about the security policies, including the protocols and ciphers that each one includes, see Supported protocols and ciphers between viewers and CloudFront.

The security policies that are available depend on the values that you specify for **SSL Certificate** and **Custom SSL Client Support** (known as `CloudFrontDefaultCertificate` and `SSLSupportMethod` in the CloudFront API):

- When **SSL Certificate** is **Default CloudFront Certificate (*.cloudfront.net)** (when `CloudFrontDefaultCertificate` is `true` in the API), CloudFront automatically sets the security policy of a standard distribution to TLSv1.

  A multi-tenant distribution is different. You can choose a security policy for a multi-tenant distribution even when it uses the default CloudFront certificate, for example when each distribution tenant has its own certificate. You can choose from the same security policies as for `sni-only`, listed below. If you don't choose one, CloudFront uses TLSv1.2_2021. Distribution tenants use the security policy of their multi-tenant distribution, including distribution tenants that have their own certificate.
- When **SSL Certificate** is **Custom SSL Certificate (example.com)** *and* **Custom SSL Client Support** is **Clients that Support Server Name Indication (SNI) - (Recommended)** (when `CloudFrontDefaultCertificate` is `false` *and* `SSLSupportMethod` is `sni-only` in the API), you can choose from the following security policies:
  - TLSv1.3_2025
  - TLSv1.2_2025
  - TLSv1.2_2021
  - TLSv1.2_2019
  - TLSv1.2_2018
  - TLSv1.1_2016

- TLSv1_2016
- TLSv1

- When **SSL Certificate** is **Custom SSL Certificate (example.com)** *and* **Custom SSL Client Support** is **Legacy Clients Support** (when `CloudFrontDefaultCertificate` is `false` *and* `SSLSupportMethod` is `vip` in the API), you can choose from the following security policies:
  - TLSv1
  - SSLv3

  In this configuration, the TLSv1.3_2025, TLSv1.2_2025, TLSv1.2_2021, TLSv1.2_2019, TLSv1.2_2018, TLSv1.1_2016, and TLSv1_2016 security policies aren't available in the CloudFront console or API. If you want to use one of these security policies, you have the following options:

  - Evaluate whether your distribution needs Legacy Clients Support with dedicated IP addresses. If your viewers support server name indication (SNI), we recommend that you update your distribution's **Custom SSL Client Support** setting to **Clients that Support Server Name Indication (SNI)** (set `SSLSupportMethod` to `sni-only` in the API). This enables you to use any of the available TLS security policies, and it can also reduce your CloudFront charges.
  - If you must keep Legacy Clients Support with dedicated IP addresses, you can request one of the other TLS security policies (TLSv1.3_2025, TLSv1.2_2025, TLSv1.2_2021, TLSv1.2_2019, TLSv1.2_2018, TLSv1.1_2016, or TLSv1_2016) by creating a case in the AWS Support Center.

    Note

    Before you contact AWS Support to request this change, consider the following:

    - When you add one of these security policies (TLSv1.3_2025, TLSv1.2_2025, TLSv1.2_2021, TLSv1.2_2019, TLSv1.2_2018, TLSv1.1_2016, or TLSv1_2016) to a Legacy Clients Support distribution, the security policy is applied to *all* non-SNI viewer requests for *all* Legacy Clients Support distributions in your AWS account. However, when viewers send SNI requests to a distribution with Legacy Clients Support, the security policy of that distribution applies. To make sure that your desired security policy is applied to *all* viewer requests sent to *all* Legacy Clients Support distributions in your AWS account, add the desired security policy to each distribution individually.
    - By definition, the new security policy doesn't support the same ciphers and protocols as the old one. For example, if you chose to upgrade a distribution's security policy

from TLSv1 to TLSv1.1_2016, that distribution will no longer support the DES-CBC3-SHA cipher. For more information about the ciphers and protocols that each security policy supports, see Supported protocols and ciphers between viewers and CloudFront.

Supported HTTP versions

Choose the HTTP versions that you want your distribution to support when viewers communicate with CloudFront.

For viewers and CloudFront to use HTTP/2, viewers must support TLSv1.2 or later, and Server Name Indication (SNI).

For viewers and CloudFront to use HTTP/3, viewers must support TLSv1.3 and Server Name Indication (SNI). CloudFront supports HTTP/3 connection migration to allow the viewer to switch networks without losing connection. For more information about connection migration, see Connection Migration at RFC 9000.

Note

For more information about supported TLSv1.3 ciphers, see Supported protocols and ciphers between viewers and CloudFront.

Note

If you use Amazon Route 53, you can use HTTPS records to allow protocol negotiation as part of the DNS lookup if the client supports it. For more information, see Create alias resource record set.

Default root object

Optional. The object that you want CloudFront to request from your origin (for example, `index.html`) when a viewer requests the root URL of your distribution (`https://www.example.com/`) instead of an object in your distribution (`https://www.example.com/product-description.html`). Specifying a default root object avoids exposing the contents of your distribution.

The maximum length of the name is 255 characters. The name can contain any of the following characters:

- A-Z, a-z
- 0-9
- _ - . * $ / ~ " '
- &, passed and returned as `&`

When you specify the default root object, enter only the object name, for example, `index.html`. Do not add a / before the object name.

For more information, see Specify a default root object.

Standard logging

Specify if you want CloudFront to log information about each request for an object and store the log files. You can enable or disable logging at any time. There is no extra charge if you enable logging, but you may accrue charges for storing and accessing the files. You can delete the logs at any time.

CloudFront supports the following standard logging options:

- Standard logging (v2) – You can send logs to delivery destinations, including Amazon CloudWatch Logs, Amazon Data Firehose, and Amazon Simple Storage Service (Amazon S3).
- Standard logging (legacy) – You can only send logs to an Amazon S3 bucket.

Connection logs

CloudFront provides connection logs that capture attributes about the TLS connections to your distributions. CloudFront generates connection logs only for connections to distributions that use mutual authentication, and for HTTPS connections to distribution tenants on the Anycast static IP addresses of their connection group. Connection logs contain information such as the client IP address and port, client certificate information, connection results, and TLS ciphers being used. These connection logs can then be used to review request patterns and other trends.

To learn more about connection logs, see Connection logs.

Log prefix

(Optional) If you enable standard logging (legacy), specify the string, if any, that you want CloudFront to prefix to the access log file names for this distribution, for example, `exampleprefix/`. The trailing slash (/) is optional but recommended to simplify browsing your log files. For more information, see Configure standard logging (legacy).

Cookie logging

If you want CloudFront to include cookies in access logs, choose **On**. If you choose to include cookies in logs, CloudFront logs all cookies regardless of how you configure the cache behaviors for this distribution: forward all cookies, forward no cookies, or forward a specified list of cookies to the origin.

Amazon S3 doesn't process cookies, so unless your distribution also includes an Amazon EC2 or other custom origin, we recommend that you choose **Off** for the value of **Cookie Logging**.

For more information about cookies, see Cache content based on cookies.

Enable IPv6 (viewer requests)

If you want CloudFront to respond to viewer requests from IPv4 and IPv6 IP addresses, select **Enable IPv6**. For more information, see Enable IPv6 for CloudFront distributions.

Mutual authentication

Optional. You can choose to turn on mutual authentication for your CloudFront distribution. For more information, see Mutual TLS authentication with CloudFront (Viewer mTLS).

Enable IPv6 for custom origins (origin requests)

When you use a custom origin (excluding Amazon S3 and VPC origins), you can customize origin settings for your distribution to choose how CloudFront connects to your origin using IPv4 or IPv6 addresses. For more information, see Enable IPv6 for CloudFront distributions.

Comment

Optional. When you create a distribution, you can include a comment of up to 128 characters. You can update the comment at any time.

Distribution state

Indicates whether you want the distribution to be enabled or disabled after it's deployed:

- *Enabled* means that as soon as the distribution is fully deployed you can deploy links that use the distribution's domain name and users can retrieve content. Whenever a distribution is enabled, CloudFront accepts and handles any end-user requests for content that use the domain name associated with that distribution.

  When you create, modify, or delete a CloudFront distribution, it takes time for your changes to propagate to the CloudFront database. An immediate request for information about a distribution might not show the change. Propagation usually completes within minutes, but a high system load or network partition might increase this time.
- *Disabled* means that even though the distribution might be deployed and ready to use, users can't use it. Whenever a distribution is disabled, CloudFront doesn't accept any end-user requests that use the domain name associated with that distribution. Until you switch the distribution from disabled to enabled (by updating the distribution's configuration), no one can use it.

You can toggle a distribution between disabled and enabled as often as you want. Follow the process for updating a distribution's configuration. For more information, see Update a distribution.

Custom error pages and error caching

You can have CloudFront return an object to the viewer (for example, an HTML file) when your Amazon S3 or custom origin returns an HTTP 4xx or 5xx status code to CloudFront. You can also specify how long an error response from your origin or a custom error page is cached in CloudFront edge caches. For more information, see Create a custom error page for specific HTTP status codes.

> **Note**
>
> The following values aren't included in the Create Distribution wizard, so you can configure custom error pages only when you update a distribution.

Topics

- HTTP error code
- Response page path
- HTTP response code

- Error caching minimum TTL (seconds)

HTTP error code

The HTTP status code for which you want CloudFront to return a custom error page. You can configure CloudFront to return custom error pages for none, some, or all of the HTTP status codes that CloudFront caches.

Response page path

The path to the custom error page (for example, `/4xx-errors/403-forbidden.html`) that you want CloudFront to return to a viewer when your origin returns the HTTP status code that you specified for **Error Code** (for example, 403). If you want to store your objects and your custom error pages in different locations, your distribution must include a cache behavior for which the following is true:

- The value of **Path Pattern** matches the path to your custom error messages. For example, suppose you saved custom error pages for 4xx errors in an Amazon S3 bucket in a directory named `/4xx-errors`. Your distribution must include a cache behavior for which the path pattern routes requests for your custom error pages to that location, for example, **/4xx-errors/***.
- The value of **Origin** specifies the value of **Origin ID** for the origin that contains your custom error pages.

HTTP response code

The HTTP status code that you want CloudFront to return to the viewer along with the custom error page.

Error caching minimum TTL (seconds)

The minimum amount of time that you want CloudFront to cache error responses from your origin server.

Geographic restrictions

If you need to prevent users in selected countries from accessing your content, you can configure your CloudFront distribution with an **Allowlist** or a **Block list**. There is no additional charge for configuring geographic restrictions. For more information, see Restrict the geographic distribution of your content.

Test a distribution

After you've created your distribution, CloudFront knows where your origin server is, and you know the domain name associated with the distribution. To test your distribution, do the following:

1. Wait until your distribution is deployed.
   - View your distribution **Details** in the console. When your distribution is done deploying, the **Last modified** field changes from **Deploying** to a date and time.
2. Create links to your objects with the CloudFront domain name by using the following procedure.
3. Test the links. CloudFront serves the objects to your webpage or application.

Create links to your objects

Use the following procedure to create test links for the objects in your CloudFront web distribution.

To create links to objects in a web distribution

1. Copy the following HTML code into a new file, replace *domain-name* with your distribution's domain name, and replace *object-name* with the name of your object.

   ```
   <html>
   <head>
    <title>My CloudFront Test</title>
   </head>
   <body>
    <p>My text content goes here.</p>
    <p><img src="https://domain-name/object-name" alt="my test image"></p>
   </body>
   </html>
   ```

   For example, if your domain name were `d111111abcdef8.cloudfront.net` and your object were `image.jpg`, the URL for the link would be:

   `https://d111111abcdef8.cloudfront.net/image.jpg`.

   If your object is in a folder on your origin server, then the folder must also be included in the URL. For example, if image.jpg were located in the images folder on your origin server, then the URL would be:

```
https://d111111abcdef8.cloudfront.net/images/image.jpg
```

2. Save the HTML code in a file that has an .html file name extension.
3. Open your webpage in a browser to make sure that you can see your object.

The browser returns your page with the embedded image file, served from the edge location that CloudFront determined was appropriate to serve the object.

Update a distribution

In the CloudFront console, you can see the CloudFront distributions that are associated with your AWS account, view the settings for a distribution, and update most settings. Be aware that settings changes that you make won't take effect until the distribution has propagated to the AWS edge locations.

Update a distribution in the console

The following procedures show you how to update a CloudFront distribution in the console.

Multi-tenant

To update a multi-tenant distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Search for and choose the ID of the multi-tenant distribution.
3. Choose the tab for the settings you want to update.
4. Make the updates, and then, to save your changes, choose **Save changes**. For more information about the settings you can update, see Preconfigured distribution settings reference.

You can also update a distribution by using the CloudFront API:

- To update a distribution, see UpdateDistribution in the *Amazon CloudFront API Reference*.

> **Important**
>
> When you update your distribution, be aware that a number of additional fields are required that are not required when you first create a distribution. To help make sure that all of the required fields are included when you use the CloudFront API to update a distribution, follow the steps described in UpdateDistribution in the *Amazon CloudFront API Reference*.

To change the multi-tenant distribution for a distribution tenant, you update the distribution tenant. You also update the distribution tenant to update its domain, certificate, customizations, or parameter values. For more details about updating the distribution tenant certificate, see Add a domain and certificate (distribution tenant).

To update a distribution tenant

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Under **SaaS**, choose **Distribution tenants**.
3. Search for the distribution tenant. Use the dropdown menu in the search bar to filter by domain, name, distribution ID, certificate ID, connection group ID, or web ACL ID.
4. Choose the distribution tenant name.
5. To update general **Details**, choose **Edit**, make the updates, and choose **Update distribution tenant**.
6. Choose the appropriate tab for any other settings to update, make your updates, and save them. For more information about the distribution tenant settings you can customize, see Distribution tenant customizations.

Standard

To update a standard distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Select the ID of a distribution. The list includes all of the distributions associated with the AWS account that you used to sign in to the CloudFront console.

3. To update general settings, choose **Edit**. Otherwise, choose the tab for the settings that you want to update.
4. Make the updates, and then choose **Save changes**. For information about the fields, see the following topics:
   - **General settings:** Distribution settings
   - **Origin settings:** Origin settings
   - **Cache behavior settings:** Cache behavior settings
5. If you want to delete an origin in your distribution, do the following:
   a. Choose **Behaviors**, and then make sure you have moved any default cache behaviors associated with the origin to another origin.
   b. Choose **Origins**, and then select an origin.
   c. Choose **Delete**.

You can also update a distribution by using the CloudFront API:

- To update a distribution, see UpdateDistribution in the *Amazon CloudFront API Reference*.

> **Important**
>
> When you update your distribution, be aware that a number of additional fields are required that are not required to create a distribution. To help make sure that all of the required fields are included when you use the CloudFront API to update a distribution, follow the steps described in UpdateDistribution in the *Amazon CloudFront API Reference*.

When you save changes to your distribution configuration, CloudFront starts to propagate the changes to all edge locations. Successive configuration changes propagate in their respective order. Until your configuration is updated in an edge location, CloudFront continues to serve your content from that location based on the previous configuration. After your configuration is updated in an edge location, CloudFront immediately starts to serve your content from that location based on the new configuration.

Your changes don't propagate to every edge location at the same time. While CloudFront is propagating your changes, we can't determine whether a given edge location is serving your content based on the previous configuration or the new configuration.

> **Note**
>
> In rare cases when a host or network link is disrupted, some distribution tenant traffic might be served using older configurations for a brief period until your changes make their way through the network.

To see when your changes are propagated, view your distribution **Details** in the console. The **Last modified** field changes from **Deploying** to a date and time when deployment is complete.

Tag a CloudFront resource

Tags are words or phrases that you can use to identify and organize your AWS resources. You can add multiple tags to each resource, and each tag includes a key and a value that you define. For example, the key might be "domain" and the value might be "example.com". You can search and filter your resources based on the tags you add. For more information about use cases for tags, see Common tagging strategies in the Tagging AWS Resources and Tag Editor Guide.

You can use tags with CloudFront, such as the following examples:

- Enforce tag-based permissions on CloudFront distributions. For more information, see ABAC with CloudFront.
- Track billing information in different categories. When you apply tags to CloudFront distributions or other AWS resources (such as Amazon EC2 instances or Amazon S3 buckets) and activate the tags, AWS generates a cost allocation report as a comma-separated value (CSV file) with your usage and costs aggregated by your active tags.

  You can apply tags that represent business categories (such as cost centers, application names, or owners) to organize your costs across multiple services. For more information about using tags for cost allocation, see Using cost allocation tags in the *AWS Billing User Guide*.

Notes

- You can tag distributions, but you can't tag origin access identities or invalidations.
- Tag Editor and Resource groups aren't currently supported for CloudFront.
- For the current maximum number of tags that you can add to a distribution, see General quotas.

Contents

- Tag restrictions
- Tag your resources for billing
- Add, edit, and delete tags
- Programmatic tagging

Tag restrictions

The following basic restrictions apply to tags:

- For the maximum number of tags per distribution, see General quotas.
- Maximum key length – 128 Unicode characters
- Maximum value length – 256 Unicode characters
- Valid values for key and value – a-z, A-Z, 0-9, space, and the following characters: _ . : / = + - and @
- Tag keys and values are case sensitive
- Don't use `aws:` as a prefix for keys. This prefix is reserved for AWS use.

Tag your resources for billing

You can use tags to organize your AWS bill to reflect your own cost structure. To do this, you must first activate tags for cost allocation in the Billing and Cost Management console. For more information about setting up a cost allocation report with tags, see Monthly cost allocation report in the AWS Billing User Guide. To see the cost of your combined resources, you can organize your billing information based on resources that have the same tag key values. For example, you can tag

several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see Using cost allocation tags in the AWS Billing User Guide.

> **Note**
>
> If you've just enabled reporting, data for the current month is available for viewing after 24 hours.

Add, edit, and delete tags

You can use the CloudFront console to manage tags for your CloudFront resources.

To add, edit, or delete tags for a resource

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Navigate to your resource:
   - **For a distribution:** Choose the ID for the distribution that you want to update.
   - **For a function:** Choose Functions, then choose the name of the function that you want to update.
   - **For a KeyValueStore:** Choose Functions, choose the KeyValueStore tab, then choose the name of the KeyValueStore that you want to update.
3. Choose the **Tags** tab.
4. Choose **Manage tags**.
5. On the **Manage tags** page, you can do the following:
   - To add a tag, enter a key and, optionally, a value for the tag. Choose **Add new tag** to add more tags.
   - To edit a tag, change the tag's key or its value, or both. You can delete the value for a tag, but the key is required.
   - To delete a tag, choose **Remove**.
6. Choose **Save changes**.

Programmatic tagging

You can also use the CloudFront API, AWS Command Line Interface (AWS CLI), AWS SDKs, and AWS Tools for Windows PowerShell to apply tags. For more information, see the following topics:

- CloudFront API operations:
  - ListTagsForResource
  - TagResource
  - UntagResource
- AWS CLI – See cloudfront in the *AWS CLI Command Reference*
- AWS SDKs – See the applicable SDK documentation on the AWS Documentation page
- Tools for Windows PowerShell – See Amazon CloudFront in the AWS Tools for PowerShell Cmdlet Reference

Delete a distribution

The following procedure deletes a distribution by using the CloudFront console. For information about deleting with the CloudFront API, see DeleteDistribution in the *Amazon CloudFront API Reference*.

If you need to delete a distribution with an OAC attached to an S3 bucket, see Delete a distribution with an OAC attached to an S3 bucket for important details.

Warning

- Before you can delete a distribution, you must disable it, which requires permission to update the distribution. Once deleted, a distribution cannot be recovered.
- If you disable a distribution that has an alternate domain name associated with it, CloudFront stops accepting traffic for that domain name (such as www.example.com), even if another distribution has an alternate domain name with a wildcard (*) that matches the same domain (such as *.example.com).
- You can't delete a distribution that is subscribed to a CloudFront flat-rate pricing plan. You will receive an error that says, "You can't delete this distribution while it's subscribed to a pricing plan." You must first cancel the pricing plan and then after the current billing cycle, delete the distribution.

Multi-tenant

Before you can delete a multi-tenant distribution, you must first delete all associated distribution tenants from it.

To delete a multi-tenant distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the right pane of the CloudFront console, choose the name of the multi-tenant distribution that you want to delete.
3. For **Tenants**, select and delete all associated distribution tenants.
4. Choose **Disable** to disable the distribution, and choose **Disable distribution** to confirm.
5. Wait until the new timestamp appears under the **Last modified** column.
   - It might take a few minutes for CloudFront to propagate your change to all edge locations.
6. Choose **Delete**, **Delete distribution**.

To delete a distribution tenant

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Under **SaaS**, choose **Distribution tenants**.
3. Search for the distribution tenant. Use the dropdown menu in the search bar to filter by domain, name, distribution ID, certificate ID, connection group ID, or web ACL ID.
4. Select the distribution tenant to delete.
5. Choose **Delete tenant**, **Delete distribution tenant**.

Standard

To delete a standard distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the right pane of the CloudFront console, find the distribution that you want to delete.

- If the **Status** column shows that the distribution is already **Disabled**, skip to Step 6.
- If the **Status** shows **Enabled** but the distribution still shows **Deploying** in the **Last modified** column, wait for deployment to finish before continuing to step 3.

3. In the right pane of the CloudFront console, select the check box for the distribution that you want to delete.
4. Choose **Disable** to disable the distribution, and choose **Yes, Disable** to confirm. Then choose **Close**.

   - The value of the **Status** column immediately changes to **Disabled**.

5. Wait until the new timestamp appears under the **Last modified** column.

   - It might take a few minutes for CloudFront to propagate your change to all edge locations.

6. Select the check box for the distribution that you want to delete.
7. Choose **Delete**, **Delete**.

   - If the **Delete** option isn't available, it means that CloudFront is still propagating your change to the edge locations. Wait until the new timestamp appears under the **Last modified** column, then repeat steps 6-7.

Use various origins with CloudFront distributions

When you create a distribution, you specify the *origin* where CloudFront sends requests for the files. You can use several different kinds of origins with CloudFront. For example, you can use an Amazon S3 bucket, a MediaStore container, a MediaPackage channel, an Application Load Balancer, or an AWS Lambda function URL. When you create your CloudFront distribution, CloudFront automatically configures most distribution settings for you, based on your content origin type. For more information, see Preconfigured distribution settings reference.

If you have an Application Load Balancer, Network Load Balancer, or EC2 instance in a private subnet, you can use it as a VPC origin. With VPC origins, your applications can be accessed only in a private subnet with a CloudFront distribution, which prevents your application from being accessible on the public internet. For more information, see the section called “Restrict access with VPC origins”.

> **Note**
>
> You can use edge functions to dynamically select the appropriate origin for each request. By using CloudFront Functions or Lambda@Edge, you can route requests to different origins based on factors such as the viewer's geographic location, the request headers, or query string parameters. For more information, see Customize at the edge with functions.

Topics

- Use an Amazon S3 bucket
- Use a MediaStore container or a MediaPackage channel
- Use an Application Load Balancer
- Use a Network Load Balancer
- Use a Lambda function URL
- Use an AWS Lambda Web Function
- Use Amazon EC2 (or another custom origin)
- Use CloudFront origin groups
- Use Amazon API Gateway

Use an Amazon S3 bucket

The following topics describe the different ways that you can use an Amazon S3 bucket as the origin for a CloudFront distribution.

Topics

- Use a standard Amazon S3 bucket
- Use Amazon S3 Object Lambda
- Use Amazon S3 Access Point
- Use an Amazon S3 bucket that's configured as a website endpoint
- Add CloudFront to an existing Amazon S3 bucket
- Move an Amazon S3 bucket to a different AWS Region
- Use an Amazon S3 Multi-Region Access Point

Use a standard Amazon S3 bucket

When you use Amazon S3 as an origin for your distribution, you place the objects that you want CloudFront to deliver in an Amazon S3 bucket. You can use any method that is supported by Amazon S3 to get your objects into Amazon S3. For example, you can use the Amazon S3 console or API, or a third-party tool. You can create a hierarchy in your bucket to store the objects, just as you would with any other standard Amazon S3 bucket.

Using an existing Amazon S3 bucket as your CloudFront origin server doesn't change the bucket in any way; you can still use it as you normally would to store and access Amazon S3 objects at the standard Amazon S3 price. You incur regular Amazon S3 charges for storing the objects in the bucket. For more information about the charges to use CloudFront, see Amazon CloudFront Pricing. For more information about using CloudFront with an existing S3 bucket, see the section called "Add CloudFront to an existing Amazon S3 bucket".

> **Important**
>
> For your bucket to work with CloudFront, the name must conform to DNS naming requirements. For more information, go to Bucket naming rules in the *Amazon Simple Storage Service User Guide*.

When you specify an Amazon S3 bucket as an origin for CloudFront, we recommend that you use the following format:

bucket-name.s3.*region*.amazonaws.com

When you specify the bucket name in this format, you can use the following CloudFront features:

- Configure CloudFront to communicate with your Amazon S3 bucket using SSL/TLS. For more information, see the section called "Use HTTPS with CloudFront".
- Use an origin access control to require that viewers access your content using CloudFront URLs, not by using Amazon S3 URLs. For more information, see the section called "Restrict access to an Amazon S3 origin".
- Update the content of your bucket by submitting POST and PUT requests to CloudFront. For more information, see the section called "HTTP methods" in the topic the section called "How CloudFront processes and forwards requests to your Amazon S3 origin".

Don't specify the bucket using the following formats:

- The Amazon S3 path style: `s3.amazonaws.com/bucket-name`
- The Amazon S3 CNAME

> **Note**
>
> CloudFront supports S3 origins using any storage class, including S3 Intelligent-Tiering. When CloudFront requests objects from an S3 origin, the objects are retrieved regardless of the storage tier they currently reside in. Using CloudFront with S3 Intelligent-Tiering doesn't impact the performance or functionality of your distribution. For more information, see Managing storage costs with Amazon S3 Intelligent-Tiering in the *Amazon Simple Storage Service User Guide*.

Use Amazon S3 Object Lambda

When you create an Object Lambda Access Point, Amazon S3 automatically generates a unique alias for your Object Lambda Access Point. You can use this alias instead of an Amazon S3 bucket name as an origin for your CloudFront distribution.

When you use an Object Lambda Access Point alias as an origin for CloudFront, we recommend that you use the following format:

alias`.s3.`*region*`.amazonaws.com`

For more information about finding the *alias*, see How to use a bucket-style alias for your S3 bucket Object Lambda Access Point in the *Amazon S3 User Guide*.

> **Important**
>
> When you use an Object Lambda Access Point as an origin for CloudFront, you must use origin access control.

For an example use case, see Use Amazon S3 Object Lambda with Amazon CloudFront to Tailor Content for End Users.

CloudFront treats an Object Lambda Access Point origin the same as a standard Amazon S3 bucket origin.

If you're using Amazon S3 Object Lambda as an origin for your distribution, you must configure the following four permissions.

Object Lambda Access Point

To add permissions for the Object Lambda Access Point

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. In the navigation pane, choose **Object Lambda Access Points**.
3. Choose the Object Lambda Access Point that you want to use.
4. Choose the **Permissions** tab.
5. Choose **Edit** in the **Object Lambda Access Point policy** section.
6. Paste the following policy into the **Policy** field.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3-object-lambda:Get*",
            "Resource": "arn:aws-eusc:s3-object-lambda:eusc-de-
east-1:123456789012:accesspoint/Object-Lambda-Access-Point-name",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:aws-
eusc:cloudfront::123456789012:distribution/CloudFront-distribution-ID"
                }
            }
        }
    ]
```

```
}
```

7. Choose **Save changes**.

Amazon S3 Access Point

To add permissions for the Amazon S3 Access Point

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. In the navigation pane, choose **Access Points**.
3. Choose the Amazon S3 Access Point that you want to use.
4. Choose the **Permissions** tab.
5. Choose **Edit** in the **Access Point policy** section.
6. Paste the following policy into the **Policy** field.

   JSON

```
{
    "Version":"2012-10-17",
    "Id": "default",
    "Statement": [
        {
            "Sid": "s3objlambda",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws-eusc:s3:eusc-de-
east-1:123456789012:accesspoint/Access-Point-name",
                "arn:aws-eusc:s3:eusc-de-
east-1:123456789012:accesspoint/Access-Point-name/object/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "s3-object-lambda.amazonaws.com"
                }
            }
```

```
        }
    ]
}
```

7. Choose **Save**.

Amazon S3 bucket

To add permissions to the Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. In the navigation pane, choose **Buckets**.
3. Choose the Amazon S3 bucket that you want to use.
4. Choose the **Permissions** tab.
5. Choose **Edit** in the **Bucket policy** section.
6. Paste the following policy into the **Policy** field.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": [
                "arn:aws-eusc:s3:::bucket-name",
                "arn:aws-eusc:s3:::bucket-name/*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:DataAccessPointAccount": "AWS-account-ID"
                }
            }
        }
    ]
```

```
}
```

7. Choose **Save changes**.

AWS Lambda function

To add permissions to the Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. In the navigation pane, choose **Functions**.
3. Choose the AWS Lambda function that you want to use.
4. Choose the **Configuration** tab, then choose **Permissions**.
5. Choose **Add permissions** in the **Resource-based policy statements** section.
6. Choose **AWS account**.
7. Enter a name for **Statement ID**.
8. Enter `cloudfront.amazonaws.com` for **Principal**.
9. Choose `lambda:InvokeFunction` from the **Action** dropdown menu.
10. Choose **Save**.

Use Amazon S3 Access Point

When you use an S3 Access Point, Amazon S3 automatically generates a unique alias for you. You can use this alias instead of an Amazon S3 bucket name as an origin for your CloudFront distribution.

When you use an Amazon S3 Access Point alias as an origin for CloudFront, we recommend that you use the following format:

alias.s3.*region*.amazonaws.com

For more information about finding the *alias*, see Using a bucket-style alias for your S3 bucket access point in the *Amazon S3 User Guide*.

Important

When you use an Amazon S3 Access Point as an origin for CloudFront, you must use origin access control.

CloudFront treats an Amazon S3 Access Point origin the same as a standard Amazon S3 bucket origin.

If you're using Amazon S3 Object Lambda as an origin for your distribution, you must configure the following two permissions.

Amazon S3 Access Point

To add permissions for the Amazon S3 Access Point

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. In the navigation pane, choose **Access Points**.
3. Choose the Amazon S3 Access Point that you want to use.
4. Choose the **Permissions** tab.
5. Choose **Edit** in the **Access Point policy** section.
6. Paste the following policy into the **Policy** field.

   JSON

```
{
    "Version":"2012-10-17",
    "Id": "default",
    "Statement": [
        {
            "Sid": "s3objlambda",
            "Effect": "Allow",
            "Principal": {"Service": "cloudfront.amazonaws.com"},
            "Action": "s3:*",
            "Resource": [
                "arn:aws-eusc:s3:eusc-de-
east-1:123456789012:accesspoint/Access-Point-name",
```

```
                "arn:aws-eusc:s3:eusc-de-
east-1:123456789012:accesspoint/Access-Point-name/object/*"
            ],
            "Condition": {
                "StringEquals": {"aws:SourceArn": "arn:aws-
eusc:cloudfront::123456789012:distribution/CloudFront-distribution-ID"}
            }
        }
    ]
}
```

7. Choose **Save**.

Amazon S3 bucket

To add permissions to the Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. In the navigation pane, choose **Buckets**.
3. Choose the Amazon S3 bucket that you want to use.
4. Choose the **Permissions** tab.
5. Choose **Edit** in the **Bucket policy** section.
6. Paste the following policy into the **Policy** field.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": [
                "arn:aws-eusc:s3:::bucket-name",
                "arn:aws-eusc:s3:::bucket-name/*"
            ],
```

```
            "Condition": {
                "StringEquals": {
                    "s3:DataAccessPointAccount": "AWS-account-ID"
                }
            }
        }
    ]
}
```

7. Choose **Save changes**.

Use an Amazon S3 bucket that's configured as a website endpoint

You can use an Amazon S3 bucket that's configured as a website endpoint as a custom origin with CloudFront. When you configure your CloudFront distribution, for the origin, enter the Amazon S3 static website hosting endpoint for your bucket. This value appears in the Amazon S3 console, on the **Properties** tab, in the **Static website hosting** pane. For example:

`http://bucket-name.s3-website-region.amazonaws.com`

For more information about specifying Amazon S3 static website endpoints, see Website endpoints in the *Amazon Simple Storage Service User Guide*.

When you specify the bucket name in this format as your origin, you can use Amazon S3 redirects and Amazon S3 custom error documents. For more information, see Configuring a custom error document and Configuring a redirect in the *Amazon Simple Storage Service User Guide*. (CloudFront also provides custom error pages. For more information, see the section called "Create a custom error page for specific HTTP status codes".)

Using an Amazon S3 bucket as your CloudFront origin server doesn't change the bucket in any way. You can still use it as you normally would and you incur regular Amazon S3 charges. For more information about the charges to use CloudFront, see Amazon CloudFront Pricing.

> **Note**
>
> If you use the CloudFront API to create your distribution with an Amazon S3 bucket that is configured as a website endpoint, you must configure it by using `CustomOriginConfig`, even though the website is hosted in an Amazon S3 bucket. For more information about creating distributions by using the CloudFront API, see CreateDistribution in the *Amazon CloudFront API Reference*.

Add CloudFront to an existing Amazon S3 bucket

If you store your objects in an Amazon S3 bucket, you can either have users get your objects directly from S3, or you can configure CloudFront to get your objects from S3 and then distribute them to your users. Using CloudFront can be more cost effective if your users access your objects frequently because, at higher usage, the price for CloudFront data transfer is lower than the price for Amazon S3 data transfer. In addition, downloads are faster with CloudFront than with Amazon S3 alone because your objects are stored closer to your users.

> **Note**
>
> If you want CloudFront to respect Amazon S3 cross-origin resource sharing settings, configure CloudFront to forward the `Origin` header to Amazon S3. For more information, see the section called “Cache content based on request headers”.

If you currently distribute content directly from your Amazon S3 bucket using your own domain name (such as example.com) instead of the domain name of your Amazon S3 bucket (such as amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com), you can add CloudFront with no disruption by using the following procedure.

To add CloudFront when you're already distributing your content from Amazon S3

1. Create a CloudFront distribution. For more information, see the section called “Create a distribution”.

   When you create the distribution, specify the name of your Amazon S3 bucket as the origin server.

   > **Important**
   >
   > For your bucket to work with CloudFront, the name must conform to DNS naming requirements. For more information, go to Bucket naming rules in the *Amazon Simple Storage Service User Guide*.

   If you're using a CNAME with Amazon S3, specify the CNAME for your distribution, too.

2. Create a test webpage that contains links to publicly readable objects in your Amazon S3 bucket, and test the links. For this initial test, use the CloudFront domain name of your distribution in the object URLs, for example, `https://d111111abcdef8.cloudfront.net/images/image.jpg`.

   For more information about the format of CloudFront URLs, see the section called “Customize file URLs”.

3. If you're using Amazon S3 CNAMEs, your application uses your domain name (for example, example.com) to reference the objects in your Amazon S3 bucket instead of using the name of your bucket (for example, amzn-s3-demo-bucket.s3.amazonaws.com). To continue using your domain name to reference objects instead of using the CloudFront domain name for your distribution (for example, d111111abcdef8.cloudfront.net), you need to update your settings with your DNS service provider.

   For Amazon S3 CNAMEs to work, your DNS service provider must have a CNAME resource record set for your domain that currently routes queries for the domain to your Amazon S3 bucket. For example, if a user requests this object:

   `https://example.com/images/image.jpg`

   The request is automatically rerouted, and the user sees this object:

   `https://amzn-s3-demo-bucket.s3.amazonaws.com/images/image.jpg`

   To route queries to your CloudFront distribution instead of your Amazon S3 bucket, you need to use the method provided by your DNS service provider to update the CNAME resource record set for your domain. This updated CNAME record redirects DNS queries from your domain to the CloudFront domain name for your distribution. For more information, see the documentation provided by your DNS service provider.

   > **Note**
   >
   > If you're using Route 53 as your DNS service, you can use either a CNAME resource record set or an alias resource record set. For information about editing resource record sets, see Editing records. For information about alias resource record sets, see Choosing between alias and non-alias records. Both topics are in the *Amazon Route 53 Developer Guide*.

For more information about using CNAMEs with CloudFront, see the section called "Use custom URLs".

After you update the CNAME resource record set, it can take up to 72 hours for the change to propagate throughout the DNS system, although it usually happens faster. During this time, some requests for your content will continue to be routed to your Amazon S3 bucket, and others will be routed to CloudFront.

Move an Amazon S3 bucket to a different AWS Region

If you're using Amazon S3 as the origin for a CloudFront distribution and you move the bucket to a different AWS Region, CloudFront can take up to an hour to update its records to use the new Region when both of the following are true:

- You're using a CloudFront origin access identity (OAI) to restrict access to the bucket.
- You move the bucket to an Amazon S3 Region that requires Signature Version 4 for authentication.

When you're using OAIs, CloudFront uses the Region (among other values) to calculate the signature that it uses to request objects from your bucket. For more information about OAIs, see the section called "Use an origin access identity (legacy, not recommended)". For a list of AWS Regions that support Signature Version 2, see Signature Version 2 signing process in the *Amazon Web Services General Reference*.

To force a faster update to CloudFront records, you can update your CloudFront distribution, for example, by updating the **Description** field on the **General** tab in the CloudFront console. When you update a distribution, CloudFront immediately checks the Region that your bucket is in. Propagation of the change to all edge locations should take only a few minutes.

Use an Amazon S3 Multi-Region Access Point

You can also use an Amazon S3 Multi-Region Access Point as a CloudFront origin. Amazon S3 Multi-Region Access Points provide a global endpoint that automatically routes requests to the closest Amazon S3 bucket based on network latency. When you use an Amazon S3 Multi-Region Access Point as an origin, specify the Multi-Region Access Point hostname as the origin domain name:

```
multi-region-access-point-alias.accesspoint.s3-global.amazonaws.com
```

To restrict access to the S3 Multi-Region Access Point so that it's only accessible through your CloudFront distribution, see the section called “Restrict access to an Amazon S3 Multi-Region Access Point origin”.

Use a MediaStore container or a MediaPackage channel

To stream video using CloudFront, you can set up an Amazon S3 bucket that is configured as a MediaStore container, or create a channel and endpoints with MediaPackage. Then you create and configure a distribution in CloudFront to stream the video.

For more information and step-by-step instructions, see the following topics:

- the section called “Serve video by using AWS Elemental MediaStore as the origin”
- the section called “Serve live video formatted with AWS Elemental MediaPackage”

Use an Application Load Balancer

You can use CloudFront to route traffic to both internal and internet-facing Application Load Balancers.

If your origin is one or more HTTP(S) servers (web servers) hosted on one or more Amazon EC2 instances, you can choose to use an internet-facing Application Load Balancer to distribute traffic to the instances. An internet-facing load balancer has a publicly resolvable DNS name and routes requests from clients to targets over the internet.

For more information about using an internet-facing Application Load Balancer as your origin for CloudFront, including how to make sure that viewers can only access your web servers through CloudFront and not by accessing the load balancer directly, see the section called “Restrict access to Application Load Balancers”.

Alternatively, you can use VPC origins to deliver content from applications that are hosted with an internal Application Load Balancer in your virtual private cloud (VPC) private subnets. VPC origins prevents your application from being accessible on the public internet. For more information, see Restrict access with VPC origins.

Use a Network Load Balancer

You can use both internal and internet-facing Network Load Balancers with Amazon CloudFront. You can use internal Network Load Balancers inside private subnets with CloudFront by using VPC

origins. CloudFront VPC origins allow you to serve content from applications hosted in private VPC subnets without exposing them to the public internet. For more information, see Restrict access with VPC origins.

Alternatively, you can also use CloudFront for delivering traffic from internet-facing Network Load Balancers. An internet-facing load balancer has a publicly resolvable DNS name and can receive requests from both clients on the internet and CloudFront distributions.

Use a Lambda function URL

A Lambda function URL is a dedicated HTTPS endpoint for a Lambda function. You can use a Lambda function URL to build a serverless web application entirely within Lambda. You can invoke the Lambda web application directly through the function URL, with no need to integrate with API Gateway or an Application Load Balancer.

If you build a serverless web application by using Lambda functions with function URLs, you can add CloudFront to get the following benefits:

- Speed up your application by caching content closer to viewers
- Use a custom domain name for your web application
- Route different URL paths to different Lambda functions using CloudFront cache behaviors
- Block specific requests using CloudFront geographic restrictions or AWS WAF (or both)
- Use AWS WAF with CloudFront to help protect your application from malicious bots, help prevent common application exploits, and enhance protection from DDoS attacks

To use a Lambda function URL as the origin for a CloudFront distribution, specify the full domain name of the Lambda function URL as the origin domain. A Lambda function URL domain name uses the following format:

function-URL-ID.lambda-url.*AWS-Region*.on.aws

When you use a Lambda function URL as the origin for a CloudFront distribution, the function URL must be publicly accessible. To do so, use one of the following options:

- If you use origin access control (OAC), the `AuthType` parameter of the Lambda function URL must use the `AWS_IAM` value and allow the `lambda:InvokeFunctionUrl` and `lambda:InvokeFunction` permissions in a resource-based policy. For more information about using Lambda function URLs for OAC, see Restrict access to an AWS Lambda function URL origin.

- If you don't use OAC, you can set the `AuthType` parameter of the function URL to `NONE` and allow the `lambda:InvokeFunctionUrl` permission in a resource-based policy.

You can also add a custom origin header to the requests that CloudFront sends to the origin, and write function code to return an error response if the header isn't present in the request. This helps to make sure that users can only access your web application through CloudFront, not directly using the Lambda function URL.

For more information about Lambda function URLs, see the following topics in the *AWS Lambda Developer Guide*:

- Lambda function URLs – A general overview of the Lambda function URLs feature
- Invoking Lambda function URLs – Includes details about the request and response payloads to use for coding your serverless web application
- Security and auth model for Lambda function URLs – Includes details about the Lambda auth types

Use an AWS Lambda Web Function

A Lambda Web Function is a web application that runs on AWS Lambda and provides its own HTTPS endpoint. The endpoint can be regional, which serves the Lambda Web Function from one AWS Region, or global, which serves it from multiple Regions. You can use a Lambda Web Function as the origin for a CloudFront distribution.

If you serve a web application from a Lambda Web Function, you can add CloudFront to get the following benefits:

- Speed up your application by caching responses closer to viewers, which also reduces the load on your Lambda Web Function
- Use a custom domain name and your own TLS certificate for your web application
- Route different URL paths to different Lambda Web Functions or other origins by using CloudFront cache behaviors
- Block specific requests using CloudFront geographic restrictions or AWS WAF (or both)
- Use AWS WAF with CloudFront to help protect your application from malicious bots, help prevent common application exploits, and enhance protection from DDoS attacks

- Use origin access control (OAC) to make sure that viewers can access your Lambda Web Function only through CloudFront

To use a Lambda Web Function as the origin for a CloudFront distribution, specify the hostname of its endpoint as the origin domain. CloudFront identifies the endpoint type from the hostname:

- Regional endpoint (`HomeRegion` and `PerRegion` endpoint types) – The hostname ends in `.lambda-web.`*`AWS-Region`*`.on.aws`
- Global endpoint (`MultiRegion` endpoint type) – The hostname ends in `.lambda-web.global.on.aws`

To use origin access control (OAC) with a Lambda Web Function, the endpoint's auth type must be `IamAuth`. The Lambda Web Function's resource-based policy must also allow the `lambda:InvokeWebFunctionEndpoint` permission. CloudFront signs requests to regional endpoints with SigV4 and to global endpoints with SigV4a. For more information, see Restrict access to an AWS Lambda Web Function origin.

For more information about Lambda Web Functions, see the following topics in the *AWS Lambda Developer Guide*:

- AWS Lambda Web Functions – A general overview of the Lambda Web Functions feature
- Lambda Web Function endpoints – Includes details about the regional and global endpoint types and their hostnames
- Endpoint access control – Includes details about the endpoint auth types and how CloudFront requests are signed

Use Amazon EC2 (or another custom origin)

You can use both internal and internet-facing EC2 instances with Amazon CloudFront. You can use internal EC2 instances inside private subnets with CloudFront by using VPC origins. CloudFront VPC origins allow you to serve content from applications hosted in private VPC subnets without exposing them to the public internet. For more information, see Restrict access with VPC origins.

A custom origin is an HTTP(S) web server with a publicly resolvable DNS name that routes requests from clients to targets over the internet. The HTTP(S) server can be hosted on AWS–for example, an Amazon EC2 instance–or hosted somewhere else. An Amazon S3 origin configured as a website

endpoint is also considered a custom origin. For more information, see the section called "Use an Amazon S3 bucket that's configured as a website endpoint".

When you use your own HTTP server as a custom origin, you specify the DNS name of the server, along with the HTTP and HTTPS ports and the protocol that you want CloudFront to use when fetching objects from your origin.

Most CloudFront features are supported when you use a custom origin with the exception of private content. Although you can use a signed URL to distribute content from a custom origin, for CloudFront to access the custom origin, the origin must remain publicly accessible. For more information, see the section called "Restrict content with signed URLs and signed cookies".

Follow these guidelines for using Amazon EC2 instances and other custom origins with CloudFront.

- Host and serve the same content on all servers that are serving content for the same CloudFront origin. For more information, see the section called "Origin settings" in the the section called "All distribution settings" topic.
- Log the `X-Amz-Cf-Id` header entries on all servers in case you need Support or CloudFront to use this value for debugging.
- Restrict requests to the HTTP and HTTPS ports that your custom origin listens on.
- Synchronize the clocks of all servers in your implementation. Note that CloudFront uses Coordinated Universal Time (UTC) for signed URLs and signed cookies, for logs, and reports. In addition, if you monitor CloudFront activity using CloudWatch metrics, note that CloudWatch also uses UTC.
- Use redundant servers to handle failures.
- For information about using a custom origin to serve private content, see the section called "Restrict access to files on custom origins".
- For information about request and response behavior and about supported HTTP status codes, see *Request and response behavior*.

If you use Amazon EC2 for a custom origin, we recommend that you do the following:

- Use an Amazon Machine Image that automatically installs the software for a web server. For more information, see the Amazon EC2 documentation.
- Use an Elastic Load Balancing load balancer to handle traffic across multiple Amazon EC2 instances and to isolate your application from changes to Amazon EC2 instances. For example,

if you use a load balancer, you can add and delete Amazon EC2 instances without changing your application. For more information, see the Elastic Load Balancing documentation.

- When you create your CloudFront distribution, specify the URL of the load balancer for the domain name of your origin server. For more information, see the section called "Create a distribution".

Use CloudFront origin groups

You can specify an origin group for your CloudFront origin if, for example, you want to configure origin failover for scenarios when you need high availability. Use origin failover to designate a primary origin for CloudFront plus a second origin that CloudFront automatically switches to when the primary origin returns specific HTTP status code failure responses.

For more information, including the steps for setting up an origin group, see the section called "Increase availability with origin failover".

Use Amazon API Gateway

You can use API Gateway as a custom origin for your CloudFront distribution. For more information, see the following topics:

- Securing Amazon API Gateway with secure ciphers using Amazon CloudFront AWS blog post
- How do I set up API Gateway with my own CloudFront distribution? AWS re:Post

Enable IPv6 for CloudFront distributions

Amazon CloudFront supports both IPv4 and IPv6 from clients to AWS edge locations. CloudFront also supports IPv6 and dual-stack (IPv4 and IPv6) connectivity towards origins. This helps you achieve end-to-end IPv6 delivery.

IPv6 is the next-generation internet protocol designed to replace IPv4. While IPv4 uses 32-bit addresses (such as 192.0.2.44), IPv6 uses 128-bit addresses (such as 2001:0db8:85a3::8a2e:0370:7334). IPv6 provides an expanded address space to accommodate more internet-connected devices.

Topics

- IPv6 viewer requests

- IPv6 origin requests

IPv6 viewer requests

In general, you should enable IPv6 if you have users on IPv6 networks who want to access your content. However, don't enable IPv6 if both of the following are true:

- You use signed URLs or signed cookies to restrict access to your content.
- You use a custom policy that includes the `IpAddress` parameter to restrict which IP addresses can access your content.

If you want to restrict access to some content by IP address and not restrict access to other content (or restrict access but not by IP address), you can create two distributions. For information about creating signed URLs by using a custom policy, see Create a signed URL using a custom policy. For information about creating signed cookies by using a custom policy, see Set signed cookies using a custom policy.

If you're using a Route 53 alias resource record set to route traffic to your CloudFront distribution, you need to create a second alias resource record set when both of the following are true:

- You enable IPv6 for the distribution
- You're using alternate domain names in the URLs for your objects

For more information, see Routing traffic to an Amazon CloudFront distribution by using your domain name in the *Amazon Route 53 Developer Guide*.

If you created a CNAME resource record set, either with Route 53 or with another DNS service, you don't need to make any changes. A CNAME record routes traffic to your distribution regardless of the IP address format of the viewer request.

If you enable IPv6 and CloudFront access logs, the `c-ip` column includes values in IPv4 and IPv6 format. For more information, see Log file fields.

> **Note**
>
> To maintain high customer availability, CloudFront responds to viewer requests by using IPv4 if our data suggests that IPv4 will provide a better user experience. To find out what

percentage of requests CloudFront is serving over IPv6, enable CloudFront logging for your distribution and parse the `c-ip` column, which contains the IP address of the viewer that made the request. This percentage should grow over time, but it will remain a minority of traffic as IPv6 is not yet supported by all viewer networks globally. Some viewer networks have excellent IPv6 support, but others don't support IPv6 at all. (A viewer network is analogous to your home internet or wireless carrier.)
For more information about our support for IPv6, see the CloudFront FAQ. For information about enabling access logs, see the Standard logging and Log prefix fields.

IPv6 origin requests

When you use a custom origin (excluding Amazon S3), you can customize origin settings for your distribution. With these settings, you choose how CloudFront connects to your origin using IPv4 or IPv6 addresses. For custom origins (excluding Amazon S3), you have the following connectivity options:

- **IPv4 only (default)** – This is the default configuration that CloudFront uses to connect to origins over IPv4.
- **IPv6 only (excluding Amazon S3 and VPC origins)** – Requires your origin domain to resolve to an IPv6 address. CloudFront exclusively uses IPv6 addresses for origin connections.
- **Dual-stack** – Enables connections over IPv4 and IPv6. CloudFront automatically chooses IPv4 or IPv6 origin connectivity to prioritize performance and availability so that you can use CloudFront as an IPv6 and IPv4 dual-stack internet gateway for your web applications.

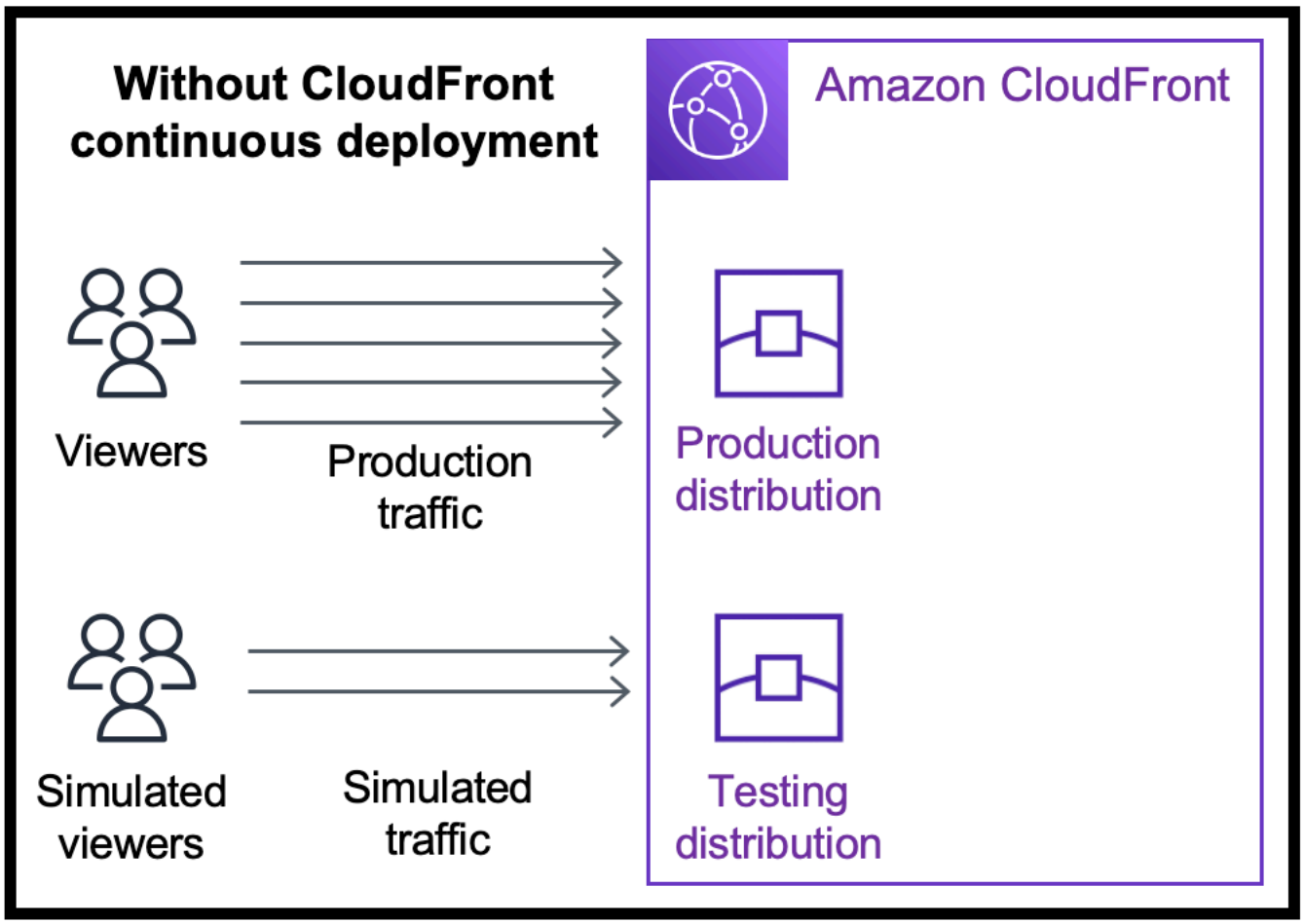
Choose the option that aligns with your origin's network configuration and connectivity requirements. For more information, see Designing DNS for IPv6 and IPv6 security and monitoring considerations.

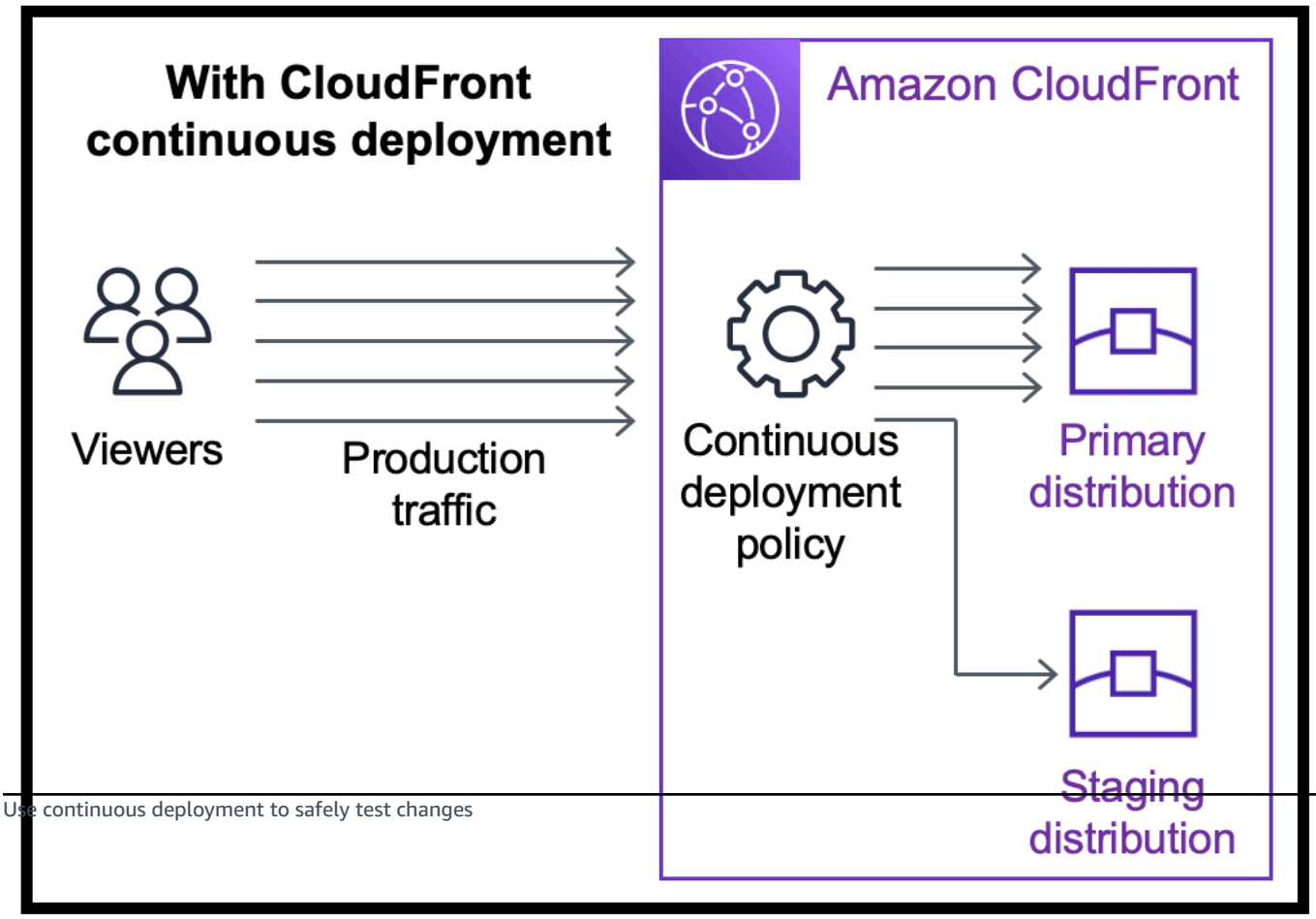
Use CloudFront continuous deployment to safely test CDN configuration changes

With Amazon CloudFront *continuous deployment* you can safely deploy changes to your CDN configuration by testing first with a subset of production traffic. You can use a *staging distribution* and a *continuous deployment policy* to send some traffic from real (production) viewers to the new

CDN configuration and validate that it works as expected. You can monitor the performance of the new configuration in real time, and promote the new configuration to serve all traffic by using the *primary distribution* when you're ready.

The following diagram shows the benefit of using CloudFront continuous deployment. Without it, you would have to test CDN configuration changes with simulated traffic. With continuous deployment you can test the changes with a subset of production traffic, then promote the changes to the primary distribution when you're ready.


Without CloudFront continuous deployment
Amazon CloudFront
Viewers
Production traffic
Production distribution
Simulated viewers
Simulated traffic
Testing distribution


With CloudFront continuous deployment
Amazon CloudFront
Viewers
Production traffic
Continuous deployment policy
Primary distribution
Staging distribution

Learn more about working with continuous deployment in the following topics.

Topics

- CloudFront continuous deployment workflow
- Work with a staging distribution and continuous deployment policy
- Monitor a staging distribution
- Learn how continuous deployment works
- Quotas and other considerations for continuous deployment

CloudFront continuous deployment workflow

The following high-level workflow explains how to safely test and deploy configuration changes with CloudFront continuous deployment.

1. Choose the distribution that you want to use as the *primary distribution*. The primary distribution is one that currently serves production traffic.
2. From the primary distribution, create a *staging distribution*. A staging distribution starts as a copy of the primary distribution.
3. Create a *traffic configuration* inside a *continuous deployment policy*, and attach it to the primary distribution. This determines how CloudFront routes traffic to the staging distribution. For more information about routing requests to a staging distribution, see the section called “Route requests to the staging distribution”.
4. Update the configuration of the staging distribution. For more information about the settings that you can update, see the section called “Update primary and staging distributions”.
5. Monitor the staging distribution to determine whether the configuration changes perform as expected. For more information about monitoring a staging distribution, see the section called “Monitor a staging distribution”.

   As you monitor the staging distribution, you can:

   - Update the configuration of the staging distribution again, to continue testing configuration changes.
   - Update the continuous deployment policy (traffic configuration) to send more or less traffic to the staging distribution.
6. When you're satisfied with the performance of the staging distribution, *promote* the staging distribution's configuration to the primary distribution, which copies the staging distribution's

configuration to the primary distribution. This also disables the continuous deployment policy which means that CloudFront routes all traffic to the primary distribution.

You can build automation that monitors the performance of the staging distribution (step 5) and promotes the configuration automatically (step 6) when certain criteria are met.

After you promote a configuration, you can reuse the same staging distribution the next time you want to test a configuration change.

For more information about working with staging distributions and continuous deployment policies in the CloudFront console, the AWS CLI, or the CloudFront API, see the following section.

Work with a staging distribution and continuous deployment policy

You can create, update, and modify staging distributions and continuous deployment policies in the CloudFront console, with the AWS Command Line Interface (AWS CLI), or with the CloudFront API.

Create a staging distribution with a continuous deployment policy

The following procedures show you how to create a staging distribution with a continuous deployment policy.

Console

You can create a staging distribution with a continuous deployment policy by using the AWS Management Console.

To create a staging distribution and continuous deployment policy (console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose the distribution that you want to use as the *primary distribution*. The primary distribution is one that currently serves production traffic, the one from which you will create the staging distribution.
4. In the **Continuous deployment** section, choose **Create staging distribution**. This opens the **Create staging distribution** wizard.

5. In the **Create staging distribution** wizard, do the following:

   a. (Optional) Type a description for the staging distribution.

   b. Choose **Next**.

   c. Modify the configuration of the staging distribution. For more information about the settings that you can update, see the section called “Update primary and staging distributions”.

      When you are finished modifying the staging distribution's configuration, choose **Next**.

   d. Use the console to specify the **Traffic configuration**. This determines how CloudFront routes traffic to the staging distribution. (CloudFront stores the traffic configuration in a *continuous deployment policy*.)

      For more information about the options in a **Traffic configuration**, see the section called “Route requests to the staging distribution”.

      When you are finished with the **Traffic configuration**, choose **Next**.

   e. Review the configuration for the staging distribution, including the traffic configuration, then choose **Create staging distribution**.

When you finish the **Create staging distribution** wizard in the CloudFront console, CloudFront does the following:

- Creates a staging distribution with the settings that you specified (in step 5c)
- Creates a continuous deployment policy with the traffic configuration that you specified (in step 5d)
- Attaches the continuous deployment policy to the primary distribution that you created the staging distribution from

When the primary distribution's configuration, with the attached continuous deployment policy, deploys to edge locations, CloudFront begins sending the specified portion of traffic to the staging distribution based on the traffic configuration.

CLI

To create a staging distribution and a continuous deployment policy with the AWS CLI, use the following procedures.

To create a staging distribution (CLI)

1. Use the **aws cloudfront get-distribution** and **grep** commands together to get the ETag value of the distribution that you want to use as the *primary distribution*. The primary distribution is one that currently serves production traffic, from which you will create the staging distribution.

   The following command shows an example. In the following example, replace *primary_distribution_ID* with the ID of the primary distribution.

   ```
   aws cloudfront get-distribution --id primary_distribution_ID | grep 'ETag'
   ```

   Copy the ETag value because you need it for the following step.

2. Use the **aws cloudfront copy-distribution** command to create a staging distribution. The following example command uses escape characters (\) and line breaks for readability, but you should omit these from the command. In the following example command:

   - Replace *primary_distribution_ID* with the ID of the primary distribution.
   - Replace *primary_distribution_ETag* with the ETag value of the primary distribution (that you got in the previous step).
   - (Optional) Replace *CLI_example* with the desired caller reference ID.

   ```
   aws cloudfront copy-distribution --primary-distribution-
   id primary_distribution_ID \
                                    --if-match primary_distribution_ETag \
                                    --staging \
                                    --caller-reference 'CLI_example'
   ```

   The command's output shows information about the staging distribution and its configuration. Copy the staging distribution's CloudFront domain name because you need it for a following step.

To create a continuous deployment policy (CLI with input file)

1. Use the following command to create file named `continuous-deployment-policy.yaml` that contains all of the input parameters for the **create-continuous-deployment-policy** command. The following command uses escape characters (\) and line breaks for readability, but you should omit these from the command.

```
aws cloudfront create-continuous-deployment-policy --generate-cli-skeleton yaml-input \
                                                    > continuous-deployment-policy.yaml
```

2. Open the file named `continuous-deployment-policy.yaml` that you just created. Edit the file to specify the continuous deployment policy settings that you want, then save the file. When you edit the file:

   - In the `StagingDistributionDnsNames` section:
     - Change the value of `Quantity` to `1`.
     - For `Items`, paste the CloudFront domain name of the staging distribution (that you saved from a previous step).
   - In the `TrafficConfig` section:
     - Choose a `Type`, either `SingleWeight` or `SingleHeader`.
     - Remove the settings for the other type. For example, if you want a weight-based traffic configuration, set `Type` to `SingleWeight` and then remove the `SingleHeaderConfig` settings.
     - To use a weight-based traffic configuration, set the value of `Weight` to a decimal number between `.01` (one percent) and `.15` (fifteen percent).

   For more information about the options in `TrafficConfig`, see the section called "Route requests to the staging distribution" and the section called "Session stickiness for weight-based configurations".

3. Use the following command to create the continuous deployment policy using input parameters from the `continuous-deployment-policy.yaml` file.

```
aws cloudfront create-continuous-deployment-policy --cli-input-yaml file://
continuous-deployment-policy.yaml
```

Copy the `Id` value in the command's output. This is the continuous deployment policy ID, and you need it in a following step.

To attach a continuous deployment policy to a primary distribution (CLI with input file)

1. Use the following command to save the primary distribution's configuration to a file named `primary-distribution.yaml`. Replace *`primary_distribution_ID`* with the primary distribution's ID.

   ```
   aws cloudfront get-distribution-config --id primary_distribution_ID --output
    yaml > primary-distribution.yaml
   ```

2. Open the file named `primary-distribution.yaml` that you just created. Edit the file, making the following changes:

   - Paste the continuous deployment policy ID (that you copied from a previous step) into the `ContinuousDeploymentPolicyId` field.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the primary distribution to use the continuous deployment policy. Replace *`primary_distribution_ID`* with the primary distribution's ID.

   ```
   aws cloudfront update-distribution --id primary_distribution_ID --cli-input-yaml
    file://primary-distribution.yaml
   ```

When the primary distribution's configuration, with the attached continuous deployment policy, deploys to edge locations, CloudFront begins sending the specified portion of traffic to the staging distribution based on the traffic configuration.

API

To create a staging distribution and continuous deployment policy with the CloudFront API, use the following API operations:

- CopyDistribution
- CreateContinuousDeploymentPolicy

For more information about the fields that you specify in these API calls, see the following:

- the section called “Route requests to the staging distribution”
- the section called “Session stickiness for weight-based configurations”
- The API reference documentation for your AWS SDK or other API client

After you create a staging distribution and a continuous deployment policy, use UpdateDistribution (on the primary distribution) to attach the continuous deployment policy to the primary distribution.

Update a staging distribution

The following procedures show you how to update a staging distribution with a continuous deployment policy.

Console

You can update certain configurations for both the primary and staging distributions. For more information, see Update primary and staging distributions.

To update a staging distribution (console)

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose the primary distribution. This is the distribution that currently serves production traffic, the one from which you created the staging distribution.
4. Choose **View staging distribution**.

5. Use the console to modify the configuration of the staging distribution. For more information about the settings that you can update, see the section called “Update primary and staging distributions”.

As soon as the staging distribution's configuration deploys to edge locations it takes effect for incoming traffic that's routed to the staging distribution.

CLI

To update a staging distribution (CLI with input file)

1. Use the following command to save the staging distribution's configuration to a file named `staging-distribution.yaml`. Replace *`staging_distribution_ID`* with the staging distribution's ID.

```
aws cloudfront get-distribution-config --id staging_distribution_ID --output
 yaml > staging-distribution.yaml
```

2. Open the file named `staging-distribution.yaml` that you just created. Edit the file, making the following changes:

   - Modify the configuration of the staging distribution. For more information about the settings that you can update, see the section called “Update primary and staging distributions”.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the staging distribution's configuration. Replace *`staging_distribution_ID`* with the staging distribution's ID.

```
aws cloudfront update-distribution --id staging_distribution_ID --cli-input-yaml
 file://staging-distribution.yaml
```

As soon as the staging distribution's configuration deploys to edge locations it takes effect for incoming traffic that's routed to the staging distribution.

API

To update the configuration of a staging distribution, use UpdateDistribution (on the staging distribution) to modify the configuration of the staging distribution. For more information about the settings that you can update, see the section called “Update primary and staging distributions”.

Update a continuous deployment policy

The following procedures show you how to update a continuous deployment policy.

Console

You can update your distribution's traffic configuration by updating the continuous deployment policy.

To update a continuous deployment policy (console)

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose the primary distribution. This is the distribution that currently serves production traffic, the one from which you created the staging distribution.
4. In the **Continuous deployment** section, choose **Edit policy**.
5. Modify the traffic configuration in the continuous deployment policy. When you are finished, choose **Save changes**.

When the primary distribution's configuration with the updated continuous deployment policy deploys to edge locations, CloudFront begins sending traffic to the staging distribution based on the updated traffic configuration.

CLI

To update a continuous deployment policy (CLI with input file)

1. Use the following command to save the continuous deployment policy's configuration to a file named `continuous-deployment-policy.yaml`. Replace *`continuous_deployment_policy_ID`* with the continuous deployment policy's ID.

The following command uses escape characters (\) and line breaks for readability, but you should omit these from the command.

```
aws cloudfront get-continuous-deployment-policy-config --
id continuous_deployment_policy_ID \
                                                    --output yaml >
 continuous-deployment-policy.yaml
```

2. Open the file named `continuous-deployment-policy.yaml` that you just created. Edit the file, making the following changes:

   - Modify the configuration of the continuous deployment policy as desired. For example, you can change from using a header-based to a weight-based traffic configuration, or you can change the percentage of traffic (weight) for a weight-based configuration. For more information, see the section called "Route requests to the staging distribution" and the section called "Session stickiness for weight-based configurations".
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the continuous deployment policy. Replace *continuous_deployment_policy_ID* with the continuous deployment policy's ID. The following command uses escape characters (\) and line breaks for readability, but you should omit these from the command.

```
aws cloudfront update-continuous-deployment-policy --
id continuous_deployment_policy_ID \
                                            --cli-input-yaml file://
continuous-deployment-policy.yaml
```

When the primary distribution's configuration with the updated continuous deployment policy deploys to edge locations, CloudFront begins sending traffic to the staging distribution based on the updated traffic configuration.

API

To update a continuous deployment policy, use UpdateContinuousDeploymentPolicy.

Promote a staging distribution configuration

The following procedures show you how to promote a staging distribution configuration.

Console

When you *promote* a staging distribution, CloudFront copies the configuration from the staging distribution to the primary distribution. CloudFront also disables the continuous deployment policy and routes all traffic to the primary distribution.

After you promote a configuration, you can reuse the same staging distribution the next time you want to test a configuration change.

To promote a staging distribution's configuration (console)

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose the primary distribution. This is the distribution that currently serves production traffic, the one from which you created the staging distribution.
4. In the **Continuous deployment** section, choose **Promote**.
5. Type **`confirm`** and then choose **Promote**.

CLI

When you *promote* a staging distribution, CloudFront copies the configuration from the staging distribution to the primary distribution. CloudFront also disables the continuous deployment policy and routes all traffic to the primary distribution.

After you promote a configuration, you can reuse the same staging distribution the next time you want to test a configuration change.

To promote a staging distribution's configuration (CLI)

- Use the **aws cloudfront update-distribution-with-staging-config** command to promote the staging distribution's configuration to the primary distribution. The following example command uses escape characters (\) and line breaks for readability, but you should omit these from the command. In the following example command:

- Replace *primary_distribution_ID* with the ID of the primary distribution.
- Replace *staging_distribution_ID* with the ID of the staging distribution.
- Replace *primary_distribution_ETag* and *staging_distribution_ETag* with the ETag values of the primary and staging distributions. Make sure the primary distribution's value is first, as shown in the example.

```
aws cloudfront update-distribution-with-staging-config --
id primary_distribution_ID \
                                                    --staging-distribution-
id staging_distribution_ID \
                                                    --if-match
 'primary_distribution_ETag,staging_distribution_ETag'
```

API

To promote a staging distribution's configuration to the primary distribution, use UpdateDistributionWithStagingConfig.

Monitor a staging distribution

To monitor the performance of a staging distribution, you can use the same metrics, logs, and reports that CloudFront provides for all distributions. For example:

- You can view the default CloudFront distribution metrics (such as total requests and error rate) in the CloudFront console, and you can turn on additional metrics (such as cache hit rate and error rate by status code) for an additional cost. You can also create alarms based on these metrics.
- You can view standard logs and real-time access logs to get detailed information about the requests that are received by the staging distribution. Standard logs contain the following two fields that help you identify the primary distribution that the request was originally sent to before CloudFront routed it to the staging distribution: `primary-distribution-id` and `primary-distribution-dns-name`.
- You can view and download reports in the CloudFront console, for example the cache statistics report.

Learn how continuous deployment works

The following topics explain how CloudFront continuous deployment works.

Topics

- Route requests to the staging distribution
- Session stickiness for weight-based configurations
- Update primary and staging distributions
- Primary and staging distributions don't share a cache

Route requests to the staging distribution

When you use CloudFront continuous deployment, you don't need to change anything about the viewer requests. Viewers cannot send requests directly to a staging distribution using a DNS name, IP address, or CNAME. Instead, viewers send requests to the primary (production) distribution, and CloudFront routes some of those requests to the staging distribution based on the traffic configuration settings in the continuous deployment policy. There are two types of traffic configurations:

Weight-based

A weight-based configuration routes the specified percentage of viewer requests to the staging distribution. When you use a weight-based configuration, you can also enable *session stickiness*, which helps make sure that CloudFront treats requests from the same viewer as part of a single session. For more information, see the section called “Session stickiness for weight-based configurations”.

Header-based

A header-based configuration routes requests to the staging distribution when the viewer request contains a specific HTTP header (you specify the header and the value). Requests that don't contain the specified header and value are routed to the primary distribution. This configuration is useful for local testing, or when you have control over the viewer requests.

> **Note**
>
> Headers routed to your staging distribution must contain the prefix `aws-cf-cd-`.

Session stickiness for weight-based configurations

When you use a weight-based configuration to route traffic to a staging distribution, you can also enable *session stickiness*, which helps make sure that CloudFront treats requests from the same viewer as a single session. When you enable session stickiness, CloudFront sets a cookie so that all requests from the same viewer in a single session are served by one distribution, either the primary or staging.

When you enable session stickiness, you can also specify the *idle duration*. If the viewer is idle (sends no requests) for this amount of time, the session expires and CloudFront treats future requests from this viewer as a new session. You specify the idle duration as a number of seconds, from 300 (five minutes) to 3600 (one hour).

In the following cases, CloudFront resets all sessions (even active ones) and considers all requests to be a new session:

- You disable or enable the continuous deployment policy
- You disable or enable the session stickiness setting

Update primary and staging distributions

When a primary distribution has an attached continuous deployment policy, the following configuration changes are available for both primary and staging distributions:

- All cache behavior settings, including the default cache behavior
- All origin settings (origins and origin groups)
- Custom error responses (error pages)
- Geographic restrictions
- Default root object
- Logging settings
- Description (comment)

You can also update external resources that are referenced in a distribution's configuration—such as a cache policy, a response headers policy, a CloudFront function, or a Lambda@Edge function.

Primary and staging distributions don't share a cache

The primary and staging distributions don't share a cache. When CloudFront sends the first request to a staging distribution, its cache is empty. As requests arrive at the staging distribution, it begins caching responses (if configured to do so).

Quotas and other considerations for continuous deployment

CloudFront continuous deployment is subject to the following quotas and other considerations.

Quotas

- Maximum number of staging distributions per AWS account: 20
- Maximum number of continuous deployment policies per AWS account: 20
- Maximum percentage of traffic you can send to a staging distribution in a weight-based configuration: 15%
- Minimum and maximum values for session stickiness idle duration: 300–3600 seconds

For more information, see *Quotas*.

> **Note**
>
> When using continuous deployment and your primary distribution is set with OAC for S3 bucket access, update your S3 bucket policy to allow access for the staging distribution. For example S3 bucket policies, see the section called “Grant CloudFront permission to access the S3 bucket”.

AWS WAF web ACLs

If you enable continuous deployment for your distribution, the following considerations apply for AWS WAF:

- You can't associate an AWS WAF web access control list (ACL) to the distribution if it's the first time that ACL has been associated with the distribution.
- You can't disassociate an AWS WAF web ACL from the distribution.

Before you can do the preceding tasks, you must delete the continuous deployment policy for your production distribution. This also deletes the staging distribution. For more information, see *Use AWS WAF protections*.

Cases when CloudFront sends all requests to the primary distribution

In certain cases, such as periods of high resource utilization, CloudFront might send all requests to the primary distribution regardless of what's specified in the continuous deployment policy.

CloudFront sends all requests to the primary distribution during peak traffic hours, regardless of what's specified in the continuous deployment policy. Peak traffic refers to the traffic on the *CloudFront service*, and not the traffic on your distribution.

HTTP/3

You cannot use continuous deployment with a distribution that supports HTTP/3.

Use custom URLs by adding alternate domain names (CNAMEs)

When you create a distribution, CloudFront provides a domain name for it, such as d111111abcdef8.cloudfront.net. Instead of using this provided domain name, you can use an alternate domain name (also known as a CNAME).

To learn how to use your own domain name, such as www.example.com, see the following topics:

Topics

- Requirements for using alternate domain names
- Restrictions on using alternate domain names
- Add an alternate domain name
- Move an alternate domain name
- Remove an alternate domain name
- Use wildcards in alternate domain names

Requirements for using alternate domain names

When you add an alternate domain name, such as www.example.com, to a CloudFront distribution, the following are requirements:

Alternate domain names must be lowercase

All alternate domain names (CNAMEs) must be lowercase.

Alternate domain names must be covered by a valid TLS certificate

To add an alternate domain name (CNAME) to a CloudFront distribution, you must attach to your distribution a trusted, valid TLS certificate that covers the alternate domain name. This makes sure that only people with access to your domain's certificate can associate with CloudFront a CNAME related to your domain.

A trusted certificate is one that is issued by AWS Certificate Manager (ACM) or by another valid certificate authority (CA). You can use a self-signed certificate to validate an existing CNAME, but *not* for a new CNAME. CloudFront supports the same certificate authorities as Mozilla. For the current list, see Mozilla Included CA Certificate List. For information about intermediate certificates when using a third-party CA, see Intermediate certificates.

To verify an alternate domain name by using the certificate that you attach, including alternate domain names that include wildcards, CloudFront checks the subject alternative name (SAN) on the certificate. The alternate domain name that you're adding must be covered by the SAN.

> **Note**
>
> Only one certificate can be attached to a CloudFront distribution at a time.

You prove that you are authorized to add a specific alternate domain name to your distribution by doing one of the following:

- Attaching a certificate that includes the alternate domain name, like product-name.example.com.
- Attaching a certificate that includes a * wildcard at the beginning of a domain name, to cover multiple subdomains with one certificate. When you specify a wildcard, you can add multiple subdomains as alternate domain names in CloudFront.

The following examples illustrate how using wildcards in domain names in a certificate work to authorize you to add specific alternate domain names in CloudFront.

- You want to add marketing.example.com as an alternate domain name. You list in your certificate the following domain name: *.example.com. When you attach this certificate to CloudFront, you can add any alternate domain name for your distribution that replaces the

wildcard at that level, including marketing.example.com. You can also, for example, add the following alternate domain names:

- product.example.com
- api.example.com

However, you can't add alternate domain names that are at levels higher or lower than the wildcard. For example, you can't add the alternate domain names example.com or marketing.product.example.com.

- You want to add example.com as an alternate domain name. To do this, you must list the domain name example.com itself on the certificate that you attach to your distribution.
- You want to add marketing.product.example.com as an alternate domain name. To do this, you can list *.product.example.com on the certificate, or you can list marketing.product.example.com itself on the certificate.

Permission to change DNS configuration

When you add alternate domain names, you must create CNAME records to route DNS queries for the alternate domain names to your CloudFront distribution. To do this, you must have permission to create CNAME records with the DNS service provider for the alternate domain names that you're using. Typically, this means that you own the domains, but you might be developing an application for the domain owner.

Alternate domain names and HTTPS

If you want viewers to use HTTPS with an alternate domain name, you must complete some additional configuration. For more information, see Use alternate domain names and HTTPS.

Restrictions on using alternate domain names

Note the following restrictions on using alternate domain names:

Maximum number of alternate domain names

For the current maximum number of alternate domain names that you can add to a distribution, or to request a higher quota (formerly known as limit), see General quotas on distributions.

Duplicate and overlapping alternate domain names

You cannot add an alternate domain name to a CloudFront distribution if the same alternate domain name already exists in another CloudFront distribution, even if your AWS account owns the other distribution.

However, you can add a wildcard alternate domain name, such as *.example.com, that includes (that overlaps with) a non-wildcard alternate domain name, such as www.example.com. If you have overlapping alternate domain names in two distributions, CloudFront sends the request to the distribution with the more specific name match, regardless of the distribution that the DNS record points to. For example, marketing.domain.com is more specific than *.domain.com.

If you have an existing wildcard DNS entry that points to a CloudFront distribution and you receive an incorrectly configured DNS error when trying to add a new CNAME with a more specific name, see CloudFront returns an incorrectly configured DNS record error when I try to add a new CNAME.

Domain fronting

CloudFront has protection against domain fronting occurring across different AWS accounts. This is a scenario when a non-standard client creates a TLS/SSL connection to a domain name in one AWS account, and then makes an HTTPS request for an unrelated domain name in another AWS account.

For example, the TLS connection might connect to www.example.com, and then issue a request for www.example.org.

To determine whether a request is being domain fronted, CloudFront performs the following checks:

- The SNI extension is equal to the HTTP request `Host` header
- The certificate belongs to the same AWS account as the distribution for the request
- The HTTP request `Host` is covered by the certificate that is served during the TLS handshake

If none of these conditions are met, CloudFront determines the request is domain fronting. CloudFront will reject the request with a 421 HTTP error response.

> **Note**
>
> If the client doesn't provide the SNI extension and gets a default *.cloudfront.net certificate instead, CloudFront will accept the incoming requests.

How CloudFront identifies the distribution for a request

CloudFront identifies a distribution for an HTTP request based on the `Host` header. CloudFront doesn't depend on the CloudFront IP address that you're connecting to or what SNI handshake was provided during TLS handshake.

When CloudFront receives a request, it will use the value of the `Host` header to match the request to the specific distribution.

For example, say that you have two distributions and you've updated your DNS configuration so that the alternate domain names route to the following endpoints:

- primary.example.com points to d111111primary.cloudfront.net
- secondary.example.com points to d222222secondary.cloudfront.net

If you make a request to https://primary.example.com but specify the `Host` header as secondary.example.com, such as `curl https://primary.example.com -H "Host: secondary.example.com"`, the request will route to the secondary distribution instead.

Adding an alternate domain name at the top node (zone apex) for a domain

When you add an alternate domain name to a distribution, you typically create a CNAME record in your DNS configuration to route DNS queries for the domain name to your CloudFront distribution. However, you can’t create a CNAME record for the top node of a DNS namespace, also known as the zone apex; the DNS protocol doesn’t allow it. For example, if you register the DNS name example.com, the zone apex is example.com. You can’t create a CNAME record for example.com, but you can create CNAME records for www.example.com, newproduct.example.com, and so on.

If you’re using Route 53 as your DNS service, you can create an alias resource record set, which has the following advantages over CNAME records:

- You can create an alias resource record set for a domain name at the top node (example.com).
- You can create an HTTPS record for an alternate domain name to allow protocol negotiation as part of the DNS lookup if the client supports it. For more information, see Create alias resource record set.
- You don’t pay for Route 53 queries when you use an alias resource record set.

> **Note**
>
> If you enable IPv6, you must create two alias resource record sets: one to route IPv4 traffic (an A record) and one to route IPv6 traffic (an AAAA record). For more information, see Enable IPv6 (viewer requests) in the topic All distribution settings reference.

For more information, see Routing traffic to an Amazon CloudFront web distribution by using your domain name in the *Amazon Route 53 Developer Guide*.

If you're not using Route 53 for your DNS, you can request Anycast static IP addresses to route apex domains like example.com to CloudFront. For more information, see Request Anycast static IPs to use for allowlisting.

Add an alternate domain name

The following task list describes how to use the CloudFront console to add an alternate domain name to your distribution so that you can use your own domain name in your links instead of the CloudFront domain name. For information about updating your distribution using the CloudFront API, see Configure distributions.

> **Note**
>
> If you want viewers to use HTTPS with your alternate domain name, see Use alternate domain names and HTTPS.

Before you begin: Make sure that you do the following before you update your distribution to add an alternate domain name:

- Register the domain name with Route 53 or another domain registrar.
- Get a TLS certificate from an authorized certificate authority (CA) that covers the domain name. Add the certificate to your distribution to validate that you are authorized to use the domain. For more information, see Requirements for using alternate domain names.

Add an alternate domain name

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the ID for the distribution that you want to update.
3. On the **General** tab, choose **Add a domain**.
4. Enter up to five domains to serve.
5. Choose **Next**.
6. For **TLS certificate**, if CloudFront can't find an existing AWS Certificate Manager (ACM) certificate for your domain in your AWS account in the `us-east-1` AWS Region, you can choose to automatically create a certificate or manually create it in ACM.
7. When your certificate is provisioned, you must update your DNS records with your DNS provider to prove domain ownership. The entries that you need to make to your DNS records are provided for you in the CloudFront console.
8. After you update your DNS records, choose **Validate certificate**.
9. When the certificate is validated, choose **Next**.
10. Review your changes and choose **Add domains**.
11. On the **General** tab for the distribution, confirm that **Distribution Status** has changed to **Deployed**. If you try to use an alternate domain name before the updates to your distribution have been deployed, the links that you create in the following steps might not work.
12. Configure the DNS service for the alternate domain name (such as www.example.com) to route traffic to the CloudFront domain name for your distribution (such as d111111abcdef8.cloudfront.net). The method that you use depends on whether you're using Route 53 as the DNS service provider for the domain or another provider. For more information, see Add a domain to your CloudFront standard distribution.

    Route 53

    Create an alias resource record set. With an alias resource record set, you don't pay for Route 53 queries. You can also create an alias resource record set for the root domain name (example.com), which DNS doesn't allow for CNAMEs. For instructions on creating an alias resource record set, see Routing traffic to an Amazon CloudFront web distribution by using your domain name in the *Amazon Route 53 Developer Guide*.

    Optionally, you can create an HTTPS record for an alternate domain name to allow protocol negotiation as part of the DNS lookup if the client supports it.

To create an alias resource record set with an HTTPS record (optional)

1. Enable HTTP/2 or HTTP/3 in your CloudFront distribution settings. For more information, see Supported HTTP versions and Update a distribution.
2. In the Route 53 console, create an alias resource record set. Follow the Routing traffic to an Amazon CloudFront web distribution by using your domain name procedure.
3. While you are creating the alias resource record set, create an alias record with record type **HTTPS**.

Another DNS service provider

Use the method provided by your DNS service provider to add a CNAME record for your domain. This new CNAME record will redirect DNS queries from your alternate domain name (for example, www.example.com) to the CloudFront domain name for your distribution (for example, d111111abcdef8.cloudfront.net). For more information, see the documentation provided by your DNS service provider.

> **Important**
>
> If you already have an existing CNAME record for your alternate domain name, update that record or replace it with a new one that points to the CloudFront domain name for your distribution.

13. Using `dig` or a similar DNS tool, confirm that the DNS configuration that you created in the previous step points to the domain name for your distribution.

    The following example shows a `dig` request on the www.example.com domain, as well as the relevant part of the response.

```
PROMPT> dig www.example.com

; <<> DiG 9.3.3rc2 <<> www.example.com
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15917
;; flags: qr rd ra; QUERY: 1, ANSWER: 9, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
```

```
;www.example.com.      IN    A

;; ANSWER SECTION:
www.example.com. 10800 IN CNAME d111111abcdef8.cloudfront.net.
...
```

The answer section shows a CNAME record that routes queries for www.example.com to the CloudFront distribution domain name d111111abcdef8.cloudfront.net. If the name on the right side of CNAME is the domain name for your CloudFront distribution, the CNAME record is configured correctly. If it's any other value, for example, the domain name for your Amazon S3 bucket, then the CNAME record is configured incorrectly. In that case, go back to step 7 and correct the CNAME record to point to the domain name for your distribution.

14. Test the alternate domain name by visiting URLs with your domain name instead of the CloudFront domain name for your distribution.
15. In your application, change the URLs for your objects to use your alternate domain name instead of the domain name of your CloudFront distribution.

Move an alternate domain name

If you try to add an alternate domain name to a standard distribution or distribution tenant, and the alternate domain name is already associated with a different resource, you will get an error message.

For example, you will get the `CNAMEAlreadyExists` error message (One or more of the CNAMEs you provided are already associated with a different resource) when you try to add www.example.com to a standard distribution or distribution tenant, but that alternate domain name is already associated with a different resource.

In that case, you might want to move the existing alternate domain name from one resource to another. This is the *source distribution* and the *target distribution*. You can move alternate domain names between either standard distributions and/or distribution tenants.

To move the alternate domain name, see the following topics:

Topics

- Set up the target standard distribution or distribution tenant
- Find the source standard distribution or distribution tenant

- Choose how to move the alternate domain name

Set up the target standard distribution or distribution tenant

Before you can move an alternate domain name, you must set up the target resource. This is the target standard distribution or distribution tenant that you're moving the alternate domain name to.

Standard distribution

To set up a target standard distribution

1. Request a TLS certificate. This certificate includes the alternate domain name as the Subject or Subject Alternative Domain (SAN), or a wildcard (*) that covers the alternate domain name that you're moving. If you don't have one, you can request one from AWS Certificate Manager (ACM) or from another certificate authority (CA) and import it into ACM.

   > **Note**
   >
   > You must request or import the certificate in the US East (N. Virginia) (`us-east-1`) Region.

   For more information, see Request a public certificate using the console and Import a certificate in the AWS Certificate Manager in the *AWS Certificate Manager User Guide*.
2. If you haven't created the target standard distribution, create one now. As part of creating the standard distribution, associate the certificate with this standard distribution. For more information, see Create a distribution.

   If you already have a target standard distribution, associate the certificate with the standard distribution. For more information, see Update a distribution.
3. **If you're using update-domain-association to move alternate domain names within the same AWS account, skip this step.**

   To move an alternate domain name from one AWS account to another, a DNS record must point the alternate domain name to the target standard distribution. This verification step

helps prevent unauthorized domain transfers. CloudFront uses this DNS record to validate your ownership of the alternate domain name.

Usually, this is the domain's CNAME record, which points to the CloudFront domain name of the target standard distribution, such as d111111abcdef8.cloudfront.net. You can also create a TXT record instead.

An apex domain, such as `example.com`, can't have a CNAME record, and an alias record doesn't prove that you control the domain. For an apex domain, create a TXT record.

The name of the TXT record depends on the type of domain. The value is the CloudFront domain name of the target standard distribution, or the routing endpoint of the target distribution tenant's connection group, such as d111111abcdef8.cloudfront.net.

Domain type	Domain	TXT record
Subdomain	`www.example.com`	`_www.example.com TXT d111111abcdef8.cloudfront.net`
Apex (root) domain	`example.com`	`_.example.com TXT d111111abcdef8.cloudfront.net`
Wildcard domain	`*.example.com`	`_*.example.com TXT d111111abcdef8.cloudfront.net`

Distribution tenant

To set up the target distribution tenant

1. Request a TLS certificate. This certificate includes the alternate domain name as the Subject or Subject Alternative Domain (SAN), or a wildcard (*) that covers the alternate domain name that you're moving. If you don't have one, you can request one from AWS Certificate Manager (ACM) or from another certificate authority (CA) and import it into ACM.

> **Note**
>
> You must request or import the certificate in the US East (N. Virginia) (`us-east-1`) Region.

For more information, see Request a public certificate using the console and Import a certificate in the AWS Certificate Manager in the *AWS Certificate Manager User Guide*.

2. If you haven't created the target distribution tenant, create one now. As part of creating the distribution tenant, associate the certificate with the distribution tenant. For more information, see Create a distribution.

   If you already have a target distribution tenant, associate the certificate with the distribution tenant. For more information, see Add a domain and certificate (distribution tenant).

3. **If you're using update-domain-association to move alternate domain names within the same AWS account, skip this step.** If you're setting `OnDomainTakeover` to `allow`, you need a DNS record that points the domain to the target distribution tenant even within the same AWS account. For more information, see Move domains when you create or update a distribution tenant.

   To move an alternate domain name from one AWS account to another, a DNS record must point the alternate domain name to the target distribution tenant. This verification step helps prevent unauthorized domain transfers, and CloudFront uses this DNS record to validate your ownership of the alternate domain name.

   Usually, this is the domain's CNAME record, which points to the routing endpoint of the target distribution tenant's connection group. You can also create a TXT record instead.

   An apex domain, such as `example.com`, can't have a CNAME record, and an alias record that resolves the domain to the shared CloudFront IP addresses doesn't prove that you control the domain. For an apex domain, create a TXT record, or an A record that resolves the domain to the Anycast static IP addresses of the target distribution tenant's connection group.

The name of the TXT record depends on the type of domain. The value is the CloudFront domain name of the target standard distribution, or the routing endpoint of the target distribution tenant's connection group, such as d111111abcdef8.cloudfront.net.

Domain type	Domain	TXT record
Subdomain	`www.example.com`	`_www.example.com TXT d111111abcdef8.cloudfront.net`
Apex (root) domain	`example.com`	`_.example.com TXT d111111abcdef8.cloudfront.net`
Wildcard domain	`*.example.com`	`_*.example.com TXT d111111abcdef8.cloudfront.net`

Next, see the following topic to find the source standard distribution or distribution tenant that is already associated with the alternate domain name.

Find the source standard distribution or distribution tenant

Before you can move an alternate domain name from one distribution (standard or tenant) to another, find the *source distribution*. This is the resource that the alternate domain name is already associated with. When you know the AWS account ID of both the source and target distribution resources, you can determine how to move the alternate domain name.

Notes

- We recommend that you use the ListDomainConflicts API operation, because it supports both standard distributions and distribution tenants.
- The ListConflictingAliases API operation only supports standard distributions.

Follow these examples to find the source distribution (standard or tenant).

list-domain-conflicts

> **Tip**
>
> - For a standard distribution, you must have the `cloudfront:GetDistribution` and `cloudfront:ListDomainConflicts` permissions.
> - For a distribution tenant, you must have the `cloudfront:GetDistributionTenant` and `cloudfront:ListDomainConflicts` permissions.

To use `list-domain-conflicts` to find the source standard distribution or distribution tenant

1. Use the `list-domain-conflicts` command as shown in the following example.

   a. Replace *`www.example.com`* with the domain name.

   b. For the `domain-control-validation-resource`, specify the ID of the target standard distribution or distribution tenant that you set up previously. You must have a standard distribution or distribution tenant that is associated with a certificate that covers the specified domain.

   c. Run this command using the credentials that are in the same AWS account as the target standard distribution or distribution tenant.

   Request

   This example specifies a distribution tenant.

   ```
   aws cloudfront list-domain-conflicts \
   --domain www.example.com \
   --domain-control-validation-resource
    "DistributionTenantId=dt_2x9GhoK0TZRsohWzv1b9It8JABC"
   ```

   Response

   For each domain name in the command's output, you can see the following:

- The resource type that the domain is associated with
- The resource ID
- The AWS account ID that owns the resource

The resource ID and the account ID are partially hidden. This allows you to identify the standard distribution or distribution tenant that belongs to your account, and helps to protect the information of ones that you don't own.

```
{
    "DomainConflicts": [
        {
            "Domain": "www.example.com",
            "ResourceType": "distribution-tenant",
            "ResourceId": "***************ohWzv1b9It8JABC",
            "AccountId": "******112233"
        }
    ]
}
```

The response lists all the domain names that conflict or overlap with the one that you specified.

Example

- If you specify *tenant1.example.com*, the response includes tenant1.example.com and the overlapping wildcard alternate domain name (*.example.com if it exists).
- If you specify **.tenant1.example.com*, the response includes *.tenant1.example.com and any alternate domain names covered by that wildcard (for example, test.tenant1.example.com, dev.tenant1.example.com, and so on).

2. In the response, find the source standard distribution or distribution tenant for the alternate domain name that you're moving, and note the AWS account ID.
3. Compare the account ID of the *source* standard distribution or distribution tenant with the account ID where you created the *target* standard distribution or distribution tenant in the previous step. You can then determine whether the source and target are in the same AWS account. This helps you determine how to move the alternate domain name.

For more information, see the list-domain-conflicts command in the *AWS Command Line Interface Reference*.

list-conflicting-aliases (standard distributions only)

> **Tip**
>
> You must have the `cloudfront:GetDistribution` and `cloudfront:ListConflictingAliases` permissions on the target standard distribution.

To use `list-conflicting-aliases` to find the source standard distribution

1. Use the `list-conflicting-aliases` command as shown in the following example.

   a. Replace *`www.example.com`* with the alternate domain name, and *`EDFDVBD6EXAMPLE`* with the ID of the target standard distribution that you set up previously.

   b. Run this command using credentials that are in the same AWS account as the target standard distribution.

   Request

   This example specifies a standard distribution.

```
aws cloudfront list-conflicting-aliases \
--alias www.example.com \
--distribution-id EDFDVBD6EXAMPLE
```

   Response

   For each alternate domain name in the command's output, you can see the ID of the standard distribution that it's associated with, and the AWS account ID that owns the standard distribution. The standard distribution and account IDs are partially hidden, which allows you to identify the standard distributions and accounts that you own, and helps to protect the information of ones that you don't own.

```
{
    "ConflictingAliasesList": {
        "MaxItems": 100,
        "Quantity": 1,
        "Items": [
            {
                "Alias": "www.example.com",
                "DistributionId": "*******EXAMPLE",
                "AccountId": "******112233"
            }
        ]
    }
}
```

The response lists the alternate domain names that conflict or overlap with the one that you specified.

Example

- If you specify *www.example.com*, the response includes www.example.com and the overlapping wildcard alternate domain name (*.example.com) if it exists.
- If you specify **.example.com*, the response includes *.example.com and any alternate domain names covered by that wildcard (for example, www.example.com, test.example.com, dev.example.com, and so on).

2. Find the standard distribution for the alternate domain name that you're moving, and note the AWS account ID. Compare this account ID with the account ID where you created the target standard distribution in the previous step. You can then determine whether these two standard distributions are in the same AWS account and how to move the alternate domain name.

   For more information, see the list-conflicting-aliases command in the *AWS Command Line Interface Reference*.

Next, see the following topic to move the alternate domain name.

Choose how to move the alternate domain name

Depending on your situation, choose from the following ways to move the alternate domain name:

The target is a distribution tenant

When you create or update the target distribution tenant, set `OnDomainTakeover` to `allow`. This works whether the source is in the same AWS account or a different one, and the source doesn't need to be disabled. For more information, see Move domains when you create or update a distribution tenant.

To move a domain that's serving traffic, use **update-domain-association** instead, which checks that a certificate covers the domain. For more information, see Migrate a live domain without downtime.

The source and target distributions (standard or tenant) are in the same AWS account

Use the **update-domain-association** command in the AWS Command Line Interface (AWS CLI) to move the alternate domain name.

This command works for all same-account moves, including when the alternate domain name is an apex domain (also called a *root domain*, like example.com). For same-account moves, it doesn't require a DNS record that points to the target.

The source and target distributions (standard or tenant) are in different AWS accounts

Use the **update-domain-association** command. The source standard distribution or distribution tenant can be enabled, so viewers keep reaching the domain during the move. A DNS record must prove that you control the alternate domain name, as described in Set up the target standard distribution or distribution tenant. This works for apex domains too.

You can also use a wildcard to move the alternate domain name. For more information, see Use a wildcard to move an alternate domain name. For additional help, see Contact AWS Support to move an alternate domain name.

> **Note**
>
> We recommend that you use **update-domain-association**. You can use the **associate-alias** command, but this command only supports standard distributions. See AssociateAlias in the *Amazon CloudFront API Reference*.

update-domain-association (standard distributions and distribution tenants)

To use `update-domain-association` to move an alternate domain name

1. Use the `update-domain-association` command, as shown in the following example.

   a. Replace *`example.com`* with the alternate domain name, and specify the ID of the target standard distribution or distribution tenant.

   b. Run this command using credentials that are in the same AWS account as the target standard distribution or distribution tenant.

> **Note the following restrictions**
>
> - In addition to the `cloudfront:UpdateDomainAssociation` permission, you must have the `cloudfront:UpdateDistribution` permission to update a standard distribution. To update a distribution tenant, you must have the `cloudfront:UpdateDistributionTenant` permission.
> - If the source and target distributions (standard or tenant) are in different AWS accounts, a DNS record must point the domain to the target, as described in Set up the target standard distribution or distribution tenant. The source can be enabled.
> - If the source and target are in different AWS accounts, and the source AWS account also serves the domain with `anycast-ip-list-only`, the move fails with `InvalidAssociation`. For more information, see Serve domains on specific IP addresses with address-based selection.
> - If the target is a distribution tenant, it doesn't already serve the domain with `anycast-ip-list-only`.
> - The target distribution must be set up as described in the section called “Set up the target standard distribution or distribution tenant”.

Request

```
aws cloudfront update-domain-association \
  --domain "www.example.com" \
  --target-resource DistributionTenantId=dt_9Fd3xTZq7Hl2KABC \
```

```
  --if-match E3UN6WX5ABC123
```

Response

```
{
    "ETag": "E7Xp1Y3N9DABC",
    "Domain": "www.example.com",
    "ResourceId": "dt_9Fd3xTZq7Hl2KABC"
}
```

This command removes the alternate domain name from the source standard distribution or distribution tenant and adds it to the target standard distribution or distribution tenant.

2. After the target distribution is fully deployed, update your DNS configuration to point your domain name to the CloudFront routing endpoint. For example, your DNS record would point your alternate domain name (`www.example.com`) to the CloudFront provided domain name d111111abcdef8.cloudfront.net. If the target is a distribution tenant, specify the connection group endpoint. For more information, see Point domains to CloudFront.

associate-alias (standard distributions only)

To use `associate-alias` to move an alternate domain name

1. Use the `associate-alias` command, as shown in the following example.

   a. Replace *`www.example.com`* with the alternate domain name, and *`EDFDVBD6EXAMPLE`* with the target standard distribution ID.

   b. Run this command using credentials that are in the same AWS account as the target standard distribution.

   Note the following restrictions

   - You must have `cloudfront:AssociateAlias` and `cloudfront:UpdateDistribution` permissions on the target standard distribution.
   - If the source and target standard distribution are in the same AWS account, you must have `cloudfront:UpdateDistribution` permission on the source standard distribution.

- If the source standard distribution and target standard distribution are in different AWS accounts, you must disable the source standard distribution first.
- The target standard distribution must be set up as described in the section called “Set up the target standard distribution or distribution tenant”.
- You must create the DNS TXT record that associates the alternate domain name with the target standard distribution, even if the source and target standard distributions are in the same AWS account. Otherwise, the command fails with Invalid or missing alias DNS TXT records.

Request

```
aws cloudfront associate-alias \
--alias www.example.com \
--target-distribution-id EDFDVBD6EXAMPLE
```

This command removes the alternate domain name from the source standard distribution and moves it to the target standard distribution.

2. After the target standard distribution is fully deployed, update your DNS configuration to point the alternate domain name’s DNS record to the distribution domain name of the target standard distribution. For example, your DNS record would point your alternate domain name (`www.example.com`) to the CloudFront provided domain name d111111abcdef8.cloudfront.net.

For more information, see the associate-alias command in the *AWS CLI Command Reference*.

Migrate a live domain without downtime

To move a domain that's serving traffic, use **update-domain-association**. It moves the domain only if a certificate on the target covers it, so HTTPS keeps working through the move. Prove that you control the domain before the move, and point the domain to the target only after the move. Viewers keep using the source's DNS name until the domain has moved, and CloudFront keeps serving the domain throughout. Because the domain still points to the source, you can also roll back without changing the DNS record that viewers use.

> **Important**
>
> This procedure applies when viewers reach the domain through the shared CloudFront IP addresses. If the source or the target uses Anycast static IPs, a live migration doesn't work the same way. Requests that still arrive on the source's Anycast static IP addresses after the move might fail until viewers pick up the DNS change, so lower the TTL first and update the domain's DNS record as soon as the move completes.

To migrate a live domain

1. Lower the TTL of the domain's DNS record, so that viewers pick up the DNS change in step 6 quickly.
2. Make sure that a certificate on the target covers the domain. For a distribution tenant, this can be the distribution tenant's certificate or its multi-tenant distribution's certificate. If no certificate covers the domain, **update-domain-association** doesn't move it.
3. If the source and the target are in different AWS accounts, create a TXT record that associates the domain with the target. Leave the DNS record that viewers use, such as a CNAME or alias record, pointing to the source. The DNS check passes on a record that points to the resource that you move the domain to, and a record that points to a different CloudFront resource doesn't block the move. So the TXT record lets you move the domain to the target, and a CNAME record that viewers use lets you move it back to the source, without a DNS change in between.

   If the source and the target are in the same AWS account, skip this step. **update-domain-association** doesn't check DNS for moves within one AWS account, in either direction.

   The name of the TXT record depends on the type of domain. The value is the CloudFront domain name of the target standard distribution, or the routing endpoint of the target distribution tenant's connection group, such as d111111abcdef8.cloudfront.net.

Domain type	Domain	TXT record
Subdomain	`www.example.com`	`_www.example.com TXT d111111abcdef8.cloudfront.net`

Domain type	Domain	TXT record
Apex (root) domain	`example.com`	`_.example.com TXT d111111abcdef8.cloudfront.net`
Wildcard domain	`*.example.com`	`_*.example.com TXT d111111abcdef8.cloudfront.net`

4. Using credentials for the AWS account that owns the target, get the target's current ETag, and then move the domain. Replace the IDs, the ETag, and the domain with your own.

```
aws cloudfront get-distribution-tenant --id dt_9Fd3xTZq7Hl2KABC

aws cloudfront update-domain-association \
  --domain "www.example.com" \
  --target-resource DistributionTenantId=dt_9Fd3xTZq7Hl2KABC \
  --if-match E3UN6WX5ABC123
```

   If the target is a standard distribution, use **get-distribution** and specify `DistributionId=EDFDVBD6EXAMPLE` for `--target-resource`. When the change reaches the edge locations, CloudFront routes requests for the domain to the target, including requests that still arrive through the source's DNS name.

5. After the target is deployed, check that it serves the domain as expected. If it doesn't, move the domain back by running **update-domain-association** again with the source as the target resource, using credentials for the source's AWS account. You don't need to change DNS first. For a move between AWS accounts, both records from step 3 are already in place, so the move back passes on the CNAME record that viewers use, which points to the source, and the TXT record that points to the target doesn't block it. If you already pointed the domain to the target in step 6, point it back to the source first.

   If viewers reach the domain through an alias record, as is common for an apex domain, the alias record doesn't pass the DNS check, so the TXT record is the only proof that you control the domain. Before you move the domain back to a source in a different AWS account, update the TXT record from step 3 to associate the domain with the source.

6. In your DNS configuration, point the domain to the target. For a distribution tenant, use the routing endpoint of its connection group. For more information, see Point domains to

CloudFront. If you need to roll back after this step, first point the domain back to the source, and then move the domain back as described in step 5.

7. After the old DNS TTL has passed and you no longer need to roll back, you can delete the TXT record, if you created one, and the source's configuration for the domain.

The target's cache starts empty, so your origin receives more requests right after the move. To limit the impact, move a few domains at a time.

Move domains when you create or update a distribution tenant

When you create or update a distribution tenant, you can move domains that are already associated with another standard distribution or distribution tenant to the distribution tenant in the same request. To do this, set `OnDomainTakeover` in the CreateDistributionTenant or UpdateDistributionTenant request. Valid values are:

- `reject` (default) – If a domain in the request is already associated with another standard distribution or distribution tenant, the request fails with `CNAMEAlreadyExists`.
- `allow` – CloudFront moves each domain in the request that's already associated with another standard distribution or distribution tenant to this distribution tenant.

Unlike **update-domain-association**, which moves one domain at a time to an existing resource, this moves up to 5 domains as part of creating or updating the distribution tenant.

CloudFront removes the domains from their sources and creates or updates the distribution tenant in one operation. If CloudFront can't move one of the domains, the request fails and no domains move.

For each domain that CloudFront moves, the following must be true:

- A DNS record proves that you control the domain and points the domain to this distribution tenant. Usually, this is the domain's CNAME record. Use one of the following records:
  - A CNAME record that points the domain to the routing endpoint of the distribution tenant's connection group.
  - An A record that resolves the domain to the Anycast static IP addresses of the distribution tenant's connection group.
  - A TXT record that associates the domain with the routing endpoint of the distribution tenant's connection group.

The name of the TXT record depends on the type of domain. The value is the CloudFront domain name of the target standard distribution, or the routing endpoint of the target distribution tenant's connection group, such as d111111abcdef8.cloudfront.net.

Domain type	Domain	TXT record
Subdomain	`www.example.com`	`_www.example.com TXT d111111abcdef8.cloudfront.net`
Apex (root) domain	`example.com`	`_.example.com TXT d111111abcdef8.cloudfront.net`
Wildcard domain	`*.example.com`	`_*.example.com TXT d111111abcdef8.cloudfront.net`

An apex domain, such as `example.com`, can't have a CNAME record, and an alias record that resolves the domain to the shared CloudFront IP addresses doesn't prove that you control the domain. For an apex domain, use a TXT record, or an A record that resolves to the Anycast static IP addresses of the connection group.

CloudFront checks DNS for every domain that it moves, including when the source and the distribution tenant are in the same AWS account.

- The source isn't owned by an AWS account that CloudFront blocks from moving domains, or by another AWS service.
- If the source is in a different AWS account, that AWS account doesn't also serve the domain with `anycast-ip-list-only`. Otherwise, the request fails with `InvalidAssociation`. For more information, see Serve domains on specific IP addresses with address-based selection.

A move can remove the last domain from the source standard distribution or distribution tenant.

A certificate isn't required to move a domain. If a certificate on the distribution tenant or its multi-tenant distribution covers the domain, CloudFront serves the domain over HTTPS as soon as it moves. If no certificate covers the domain, CloudFront still moves it and serves it over HTTP only,

so HTTPS requests for the domain fail until you add a certificate. For more information, see Serve distribution tenant domains without a certificate.

If you request a CloudFront managed certificate for a domain that you're moving, CloudFront moves the domain when you make the request, and HTTPS requests for the domain fail until the certificate is issued. If you request the managed certificate with self-hosted validation and CloudFront can't verify the DNS record for a domain, the request fails and no domains move.

> **Important**
>
> CloudFront doesn't keep a record of where a moved domain came from. If you later remove the domain from the distribution tenant, delete the distribution tenant, or roll back the change, for example in a CloudFormation stack rollback, CloudFront doesn't return the domain to the source. CloudFront stops serving the domain until you add it to a resource again.
>
> To undo a move, move the domain back to the original resource instead of removing it from the distribution tenant. For more information, see Move a domain back to its original resource.

To move domains when you update a distribution tenant

1. If the source serves a domain over HTTPS, add a certificate that covers the domain to the distribution tenant or its multi-tenant distribution.

   > **Warning**
   >
   > If no certificate covers the domain, HTTPS requests for the domain start failing when it moves. To have CloudFront check this for you, move the domain with **update-domain-association** instead. For more information, see Migrate a live domain without downtime.

2. Set up the DNS record for each domain that you're moving, as described earlier in this topic. For example, create a CNAME record that points `www.example.com` to the routing endpoint of the target distribution tenant's connection group. For an apex domain, create a TXT record, or an A record that resolves the domain to the Anycast static IP addresses of the connection group.

> **Note**
>
> Until the move completes, the source continues to serve the domain.

3. Update the distribution tenant with the full list of its domains, including the domains that you're moving, and set `--on-domain-takeover` to `allow`.

```
aws cloudfront update-distribution-tenant \
  --id dt_9Fd3xTZq7Hl2KABC \
  --if-match E3UN6WX5ABC123 \
  --domains '[{"Domain": "app.example.com"}, {"Domain": "www.example.com"}]' \
  --on-domain-takeover allow
```

If the request succeeds, the response lists the moved domains in `Domains`. CloudFront doesn't return `OnDomainTakeover` in the response, and you must set it again in each request that moves domains.

To move domains while you create a distribution tenant, add `--on-domain-takeover allow` to the **create-distribution-tenant** command.

Move a domain back to its original resource

To undo a move, move the domain from the distribution tenant back to the original standard distribution or distribution tenant instead of removing it from the distribution tenant. Choose one of the following options.

update-domain-association (any original resource)

To move a domain back with `update-domain-association`

1. If the original resource is in a different AWS account than the distribution tenant, make sure that the original resource is set up as described in Set up the target standard distribution or distribution tenant, including a DNS record that proves that you control the domain. You don't need to disable the distribution tenant that has the domain now.
2. Using credentials for the AWS account that owns the original resource, get the resource's current ETag. Replace the ID with your own.

```
aws cloudfront get-distribution --id EDFDVBD6EXAMPLE
```

If the original resource is a distribution tenant, use **get-distribution-tenant** instead.

3. Move the domain to the original resource. Replace the domain, the resource ID, and the ETag with your own.

```
aws cloudfront update-domain-association \
  --domain "www.example.com" \
  --target-resource DistributionId=EDFDVBD6EXAMPLE \
  --if-match E2QWRUHEXAMPLE
```

If the original resource is a distribution tenant, specify `DistributionTenantId=dt_EXAMPLE22222` for `--target-resource`.

4. After the original resource is deployed, update the domain's DNS records to point to it. For a standard distribution, use the distribution domain name, such as d111111abcdef8.cloudfront.net. For a distribution tenant, use the routing endpoint of its connection group.

OnDomainTakeover (original resource is a distribution tenant)

If the original resource is a distribution tenant, you can also move the domain back the same way that you moved it.

To move a domain back with `OnDomainTakeover`

1. Update the domain's DNS records to point to the original distribution tenant, for example with a CNAME record to the routing endpoint of its connection group. For an apex domain, also create a TXT record that associates the domain with the original distribution tenant, unless the domain's A record resolves to the Anycast static IP addresses of its connection group. Until the move completes, the distribution tenant that has the domain now continues to serve it.
2. Using credentials for the AWS account that owns the original distribution tenant, update it with the full list of its domains, including the domain that you're moving back. Replace the ID, the ETag, and the domains with your own.

```
aws cloudfront update-distribution-tenant \
  --id dt_EXAMPLE22222 \
  --if-match E2QWRUHEXAMPLE \
  --domains '[{"Domain": "www.example.com"}]' \
```

```
  --on-domain-takeover allow
```

If you already removed the domain from the distribution tenant, no resource has the domain. Add it to the original resource again as an alternate domain name, and point the domain's DNS records back to that resource.

Use a wildcard to move an alternate domain name

If the source distribution is in a different AWS account than the target distribution, you can also use a wildcard to move the alternate domain name, instead of **update-domain-association**.

> **Note**
>
> You can't use a wildcard to move an apex domain (like example.com). To move an apex domain, use **update-domain-association**.

To use a wildcard to move an alternate domain name

> **Note**
>
> This process involves multiple updates to your distributions. Wait for each distribution to fully deploy the latest change before proceeding to the next step.

1. Update the target distribution to add a wildcard alternate domain name that covers the alternate domain name that you are moving. For example, if the alternate domain name that you're moving is www.example.com, add the alternate domain name *.example.com to the target distribution. To do this, the SSL/TLS certificate on the target distribution must include the wildcard domain name. For more information, see the section called "Update a distribution".
2. Update the DNS settings for the alternate domain name to point to the domain name of the target distribution. For example, if the alternate domain name that you're moving is www.example.com, update the DNS record for www.example.com to route traffic to the domain name of the target distribution (for example d111111abcdef8.cloudfront.net).

> **Note**
>
> Even after you update the DNS settings, the alternate domain name is still served by the source distribution because that's where the alternate domain name is currently configured.

3. Update the source distribution to remove the alternate domain name. For more information, see Update a distribution.
4. Update the target distribution to add the alternate domain name. For more information, see Update a distribution.
5. Use **dig** (or a similar DNS query tool) to validate that the DNS record for the alternate domain name resolves to the domain name of the target distribution.
6. (Optional) Update the target distribution to remove the wildcard alternate domain name.

Contact AWS Support to move an alternate domain name

If the source and target distributions are in different AWS accounts, and you can't move the alternate domain name yourself, you can contact Support to move it.

To contact Support to move an alternate domain name

1. Set up a target distribution, including the DNS TXT record that points to the target distribution. For more information, see Set up the target standard distribution or distribution tenant.
2. Contact Support to request that they verify that you own the domain, and move the domain to the new CloudFront distribution for you.
3. After the target distribution is fully deployed, update your DNS configuration to point the alternate domain name's DNS record to the distribution domain name of the target distribution.

Remove an alternate domain name

If you want to stop routing traffic for a domain or subdomain to a CloudFront distribution, follow the steps in this section to update both the DNS configuration and the CloudFront distribution.

It's important that you remove the alternate domain names from the distribution as well as update your DNS configuration. This helps prevent issues later if you want to associate the domain name with another CloudFront distribution. If an alternate domain name is already associated with one distribution, it can't be set up with another.

> **Note**
>
> If you want to remove the alternate domain name from this distribution so you can add it to another one, follow the steps in Move an alternate domain name. If you follow the steps here instead (to remove a domain) and then add the domain to another distribution, there will be a period of time during which the domain won't link to the new distribution because CloudFront is propagating the updates to edge locations.

To remove an alternate domain name from a distribution

1. To start, route internet traffic for your domain to another resource that isn't your CloudFront distribution, such as an Elastic Load Balancing load balancer. Or you can delete the DNS record that's routing traffic to CloudFront.

   Do one of the following, depending on the DNS service for your domain:

   - **If you're using Route 53**, update or delete alias records or CNAME records. For more information, see Editing records or Deleting records.
   - **If you're using another DNS service provider**, use the method provided by the DNS service provider to update or delete the CNAME record that directs traffic to CloudFront. For more information, see the documentation provided by your DNS service provider.
2. After you update your domain's DNS records, wait until the changes have propagated and DNS resolvers are routing traffic to the new resource. You can check to see when this is complete by creating some test links that use your domain in the URL.
3. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home, and update your CloudFront distribution to remove the domain name by doing the following:

   a. Choose the ID for the distribution that you want to update.

   b. On the **General** tab, choose **Edit**.

c. In **Alternate Domain Names (CNAMEs)**, remove the alternate domain name (or domain names) that you no longer want to use for your distribution.

d. Choose **Yes, Edit**.

Use wildcards in alternate domain names

When you add alternate domain names, you can use the * wildcard at the beginning of a domain name instead of adding subdomains individually. For example, with an alternate domain name of *.example.com, you can use any domain name that ends with example.com in your URLs, such as www.example.com, product-name.example.com, marketing.product-name.example.com, and so on. The path to the object is the same regardless of the domain name, for example:

- www.example.com/images/image.jpg
- product-name.example.com/images/image.jpg
- marketing.product-name.example.com/images/image.jpg

Follow these requirements for alternate domain names that include wildcards:

- The alternate domain name must begin with an asterisk and a dot (*.).
- You *cannot* use a wildcard to replace part of a subdomain name, like this: *domain.example.com.
- You cannot replace a subdomain in the middle of a domain name, like this: subdomain.*.example.com.
- All alternate domain names, including alternate domain names that use wildcards, must be covered by the subject alternative name (SAN) on the certificate.

A wildcard alternate domain name, such as *.example.com, can include another alternate domain name that's in use, such as example.com.

Use WebSockets with CloudFront distributions

Amazon CloudFront supports using WebSocket, a TCP-based protocol that is useful when you need long-lived bidirectional connections between clients and servers. A persistent connection is often a requirement with real-time applications. The scenarios in which you might use WebSockets include social chat platforms, online collaboration workspaces, multi-player gaming, and services that

provide real-time data feeds like financial trading platforms. Data over a WebSocket connection can flow in both directions for full-duplex communication.

WebSocket functionality is automatically enabled to work with any distribution. To use WebSockets, configure one of the following in the cache behavior that's attached to your distribution:

- Forward all viewer request headers to your origin. You can use the AllViewer managed origin request policy.
- Specifically forward the `Sec-WebSocket-Key` and `Sec-WebSocket-Version` request headers in your origin request policy.

How the WebSocket protocol works

The WebSocket protocol is an independent, TCP-based protocol that allows you to avoid some of the overhead—and potentially increased latency—of HTTP.

To establish a WebSocket connection, the client sends a regular HTTP request that uses HTTP's upgrade semantics to change the protocol. The server can then complete the handshake. The WebSocket connection remains open and either the client or server can send data frames to each other without having to establish new connections each time.

By default, the WebSocket protocol uses port 80 for regular WebSocket connections and port 443 for WebSocket connections over TLS. The options that you choose for your CloudFront Viewer protocol policy and Protocol (custom origins only) apply to WebSocket connections and also HTTP traffic.

WebSocket requirements

WebSocket requests must comply with RFC 6455 in the following standard formats.

Example Sample client request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: https://example.com
```

```
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Example Sample server response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

If the WebSocket connection is disconnected by the client or server, or by a network disruption, client applications are expected to re-initiate the connection with the server.

Recommended WebSocket headers

To avoid unexpected compression-related issues when using WebSockets, we recommend that you include the following headers in an origin request policy:

- `Sec-WebSocket-Key`
- `Sec-WebSocket-Version`
- `Sec-WebSocket-Protocol`
- `Sec-WebSocket-Accept`
- `Sec-WebSocket-Extensions`

HTTP versions for WebSocket connections

Viewers connect to CloudFront for WebSockets over HTTP/1.1. CloudFront doesn't support WebSocket connections from viewers over HTTP/2 or HTTP/3.

For the connection from CloudFront to your origin, CloudFront uses HTTP/1.1 by default. CloudFront can use HTTP/2 for WebSocket connections to your origin when both of the following are true:

- The origin HTTP version is HTTP/2, and the origin supports HTTP/2.
- The origin supports WebSockets over HTTP/2 with the extended `CONNECT` method (see RFC 8441 on the IETF Datatracker website). The origin advertises this with the `SETTINGS_ENABLE_CONNECT_PROTOCOL` setting.

Even when both are true, CloudFront might use HTTP/1.1 for some WebSocket connections to the origin, for example to limit resource usage. If the origin doesn't support WebSockets over HTTP/2, CloudFront uses HTTP/1.1.

Request Anycast static IPs to use for allowlisting

You can also bring your own IP address range to use with CloudFront instead of CloudFront-provided Anycast static IPs. For more information, see Bring your own IP to CloudFront using IPAM.

You can request Anycast static IPs from CloudFront to use with your distributions. Anycast static IP lists can contain only IPv4 IP addresses or both IPv4 and IPv6 IP addresses. These IP addresses are dedicated to your AWS account and spread across geographic regions.

You can request 21 Anycast static IP addresses to allowlist with network providers so that you can waive data charges for viewers who access your application. Alternatively, you can use these static IPs within outbound security firewalls to control traffic exchange with approved applications. Anycast static IP lists can be used with one or more distributions.

If you want to enable routing of apex domains (such as example.com) directly to your CloudFront distributions, you can request 3 Anycast static IP addresses for this use case. Then, add A records in your DNS to point the apex domain to CloudFront.

Anycast static IPs work with Server Name Indication (SNI). For more information, see Use SNI to serve HTTPS requests (works for most clients). To serve viewers that don't send SNI on the Anycast static IPs of a connection group, see Serve viewers that don't send SNI with a viewer fallback.

Prerequisites

To use Anycast static IP lists with your CloudFront distribution, you must select **Use all edge locations** for the price class for the distribution. For more information about pricing, see CloudFront pricing.

Request an Anycast static IP list

Request an Anycast static IP list to use with your CloudFront distribution.

To request an Anycast static IP list

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. In the left navigation pane, choose **Static IPs**.
3. For **Request**, choose the link to contact CloudFront support engineering.
4. Provide your workload information (request bytes per second and requests per second).
5. CloudFront support engineering reviews your request. The review process might take up to two days.

After your request is approved, you can create an Anycast static IP list and associate it with one or more distributions.

Create an Anycast static IP list

Before you begin, request an Anycast static IP list as explained in the preceding section.

To create an Anycast static IP list

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the left navigation pane, choose **Static IPs**.
3. Choose **Create Anycast IP list**.
4. For **Name**, enter a name.
5. For **Static IP use cases**, select the appropriate use case.
6. For **IP address type**, specify one of the following options:
   - **IPv4** – Allocate a list of only IPv4 addresses
   - **Dualstack** – Allocate a list of both IPv4 and IPv6 addresses
7. Review the service terms and pricing, and choose **Submit**.

After your static IP list is created, you can view the allocated IP addresses on your static IP list detail page. You can also associate distributions with the static IP list.

Associate an Anycast static IP list with an existing distribution

Before you begin, request and create an Anycast static IP list as explained in the preceding sections.

Verify that the following distribution settings are compatible with your Anycast static IP list:

- Price class has the **Use all edge locations (best performance)** setting.
- If IPv6 is enabled, you can associate a dualstack Anycast static IP list. An Anycast static IP list that only has IPv4 addresses can't be associated to distributions with IPv6 enabled.

To associate an Anycast static IP list with an existing distribution

- Do one of the following:
  - Associate the static IP list from the static IP list detail page:
    1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
    2. Choose **Static IPs** in the left navigation pane.
    3. Choose the name of your static IP list.
    4. Choose **Associate distributions**.
    5. Select one or more distributions and choose **Associate distributions**.
  - Associate the static IP list from the distribution detail page:
    1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
    2. Choose **Distributions** in the left navigation pane.
    3. Choose the name of your distribution.
    4. On the **General** tab, under **Settings**, choose **Edit**.
    5. For **Anycast IP list**, select the Anycast static IP list to use with this distribution.
    6. Choose **Save changes**.

Associate an Anycast static IP list with a new distribution

Before you begin, request and create an Anycast static IP list as explained in the preceding sections.

To associate an Anycast static IP list with a new distribution

- Create a new distribution. For more information, see Create a CloudFront distribution in the console. For **Settings**, you must make the following selections to use your Anycast static IP list:
  - For **Anycast IP list**, select your Anycast static IP list from the dropdown list.
  - For **Price class**, select **Use all edge locations (best performance)**.

- **Note:** If your Anycast static IP is only using IPv4 and not dualstack, for **IPv6**, select **Off**.

Finish creating your distribution. You can choose any other settings and configurations that are not required for Anycast static IP lists based on your needs.

For more information about quotas related to Anycast static IP lists, see Amazon CloudFront endpoints and quotas in the *AWS General Reference*.

Associate an Anycast static IP list with a connection group

Before you begin, request and create an Anycast static IP list as explained in the previous sections.

To associate an Anycast static IP list with a new connection group

1. Make sure you have enabled connection groups under **Settings**.
2. Create a connection group. For more information, see Create custom connection group.
3. For **Settings**, you must make the following selections to use your Anycast static IP list.
   - For **Anycast IP list**, select your Anycast static IP list from the dropdown list.
4. Finish creating your connection group.

> **Note**
>
> If your Anycast static IP is only using IPv4 and not dualstack, for **IPv6**, select **Off**.

For more information about quotas related to Anycast static IP lists, see Amazon CloudFront endpoints and quotas in the *Amazon Web Services General Reference*.

Update an Anycast static IP list

After you have created your Anycast static IP address and associated it to a distribution, you can change the IP address type of your Anycast static IP list.

To update an Anycast static IP list

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. In the left navigation pane, choose **Static IPs**.
3. Choose the name of your static IP list.
4. Choose **Edit**.
5. For **IP address type**, specify one of the following options:

   - **IPv4** – Allocate a list of only IPv4 addresses
   - **Dualstack** – Allocate a list of both IPv4 and IPv6 addresses

   > **Note**
   >
   > You can't choose **IPv4** if your associated distribution has already enabled IPv6. To do so, disable IPv6 before you can update the IP address type for your Anycast static IP. For more information, see Enable IPv6 for CloudFront distributions.

6. Choose **Submit** to save your changes and update the Anycast static IP list.

Bring your own IP to CloudFront using IPAM

This tutorial shows how to use IPAM to manage your BYOIP CIDRs for CloudFront Anycast Static IP lists.

Topics

- What is BYOIP for Anycast Static IPs?
- Why use this feature?
- Prerequisites
- Step 1: Request an Anycast static IP list
- Step 2: Create an Anycast static IP list
- Step 3: Create a CloudFront distribution
- Step 4: Associate with CloudFront resources
- Step 5: Prepare for migration
- Step 6: Advertise CIDR globally

What is BYOIP for Anycast Static IPs?

CloudFront supports bringing your own IPv4 and IPv6 addresses through IPAM's BYOIP for global services. Through IPAM's unified interface, customers can create dedicated IP address pools using their own IP addresses (BYOIP) and assign them to CloudFront distributions while leveraging the AWS worldwide content delivery network to deliver their applications and content. Your IP addresses are advertised from multiple CloudFront edge locations simultaneously using anycast routing.

Why use this feature?

Control network access in allow lists to:

- Allow-list IP addresses with network providers to waive data charges for approved viewers
- Configure outbound security firewalls to restrict traffic to approved applications only

Simplify operations and migrations

- Route apex domains (example.com) directly to CloudFront by adding A records that point to your static IPs
- Migrate from other CDNs without updating IP infrastructure or firewall configurations
- Maintain existing IP allowlists with partners and clients
- Share a single Anycast static IP list across multiple CloudFront distributions

Consistent branding

- Keep your existing IP address space for consistent branding when moving to AWS

Prerequisites

To use Anycast static IP lists with your CloudFront distribution, you must select **Use all edge locations** for the price class for the distribution. For more information about pricing, see CloudFront pricing. For Bring Your Own IP (BYOIP), you also need to disable IPv6 for the distribution or connection group when using IPv4-only BYOIP. For dual-stack BYOIP, associate a dual-stack Anycast static IP List and enable IPv6 for the distribution or connection group.

Complete these steps before starting:

- IPAM setup: See Integrate IPAM with accounts and Create an IPAM.
- Domain verification: Verify domain control.
- Create a top-level pool: Follow steps 1 to 2 in Bring your own IPv4 CIDR to IPAM and/or Bring your own IPv6 CIDR to IPAM.
- Create an IPAM pool with locale as global to use with CloudFront. For more information, see Bring your own IP to CloudFront using IPAM.

> **Note**
>
> Requires **three /24 and/or /48** IPv4 CIDR blocks.

Step 1: Request an Anycast static IP list

Request an Anycast static IP list to use with your CloudFront distribution.

To request an Anycast static IP list

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the left navigation pane, choose **Static IPs**.
3. For **Request**, choose the link to contact CloudFront support engineering.
4. Provide your workload information (request bytes per second and requests per second).
5. CloudFront support engineering reviews your request. The review process might take up to two days.
6. After your request is approved, you can create an Anycast static IP list and associate it with one or more distributions.

Step 2: Create an Anycast static IP list

Before you begin, request an Anycast static IP list as explained in the preceding section.

To create an Anycast static IP list

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. In the left navigation pane, choose **Static IPs**.
3. Choose **Create Anycast IP list**.
4. For **Name**, enter a name.
5. For **Static IP use cases**, select **BYOIP** as your use case.
6. For **IP address type**, pick **IPv4** or **Dualstack**.
7. For **IPAM pool**, pick the IPAM pool(s) you provisioned through IPAM and CIDR group(s) from them.

The following steps differ from the standard regional BYOIP process and establish the pattern for global services:

AWS CLI

Installing or updating to the latest version of the AWS CLI. For more information, see the AWS Command Line Interface User Guide.

1. Retrieve the IpamPoolArn of the IPAM pool where your CIDR blocks were provisioned. For more information, see Bring your own public IPv4 CIDR to IPAM using only the AWS CLI or Bring your own public IPv6 CIDR to IPAM using only the AWS CLI.
2. Create an Anycast IP list with your CIDR blocks and IPAM configuration:

   For IPv4:

```
aws cloudfront create-anycast-ip-list \
    --name byoip-aip-1 \
    --ip-count 3 \
    --region us-east-1 \
    --ip-address-type ipv4 \
    --ipam-cidr-configs
 '[{"Cidr":"1.1.1.0/24","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/ipam-
pool-005d58a8aa8147abc"},
{"Cidr":"2.2.2.0/24","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/ipam-
pool-005d58a8aa8147abc"},
{"Cidr":"3.3.3.0/24","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/ipam-
pool-005d58a8aa8147abc"}]'
```

   For IPv6:

```
aws cloudfront create-anycast-ip-list \
```

```
    --name byoip-aip-dualstack \
    --ip-count 3 \
    --region us-east-1 \
    --ip-address-type dualstack \
    --ipam-cidr-configs
 '[{"Cidr":"1.1.1.0/24","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/ipam-
pool-005d58a8aa8147abc"},
{"Cidr":"2.2.2.0/24","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/ipam-
pool-005d58a8aa8147abc"},
{"Cidr":"3.3.3.0/24","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/ipam-
pool-005d58a8aa8147abc"},
{"Cidr":"2600:9000:a100::/48","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/
ipam-pool-0a3b7c9e2f41d6789"},
{"Cidr":"2600:9000:a200::/48","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/
ipam-pool-0a3b7c9e2f41d6789"},
{"Cidr":"2600:9000:a300::/48","IpamPoolArn":"arn:aws:ec2::123456789012:ipam-pool/
ipam-pool-0a3b7c9e2f41d6789"}]'
```

Note

You can't select the specific IP address from the pool. CloudFront will do this automatically.

Step 3: Create a CloudFront distribution

For CloudFront, you can follow instructions to create a standard distribution or use multi-tenant distributions.

Step 4: Associate with CloudFront resources

- Associate an Anycast static IP list with an existing distribution
- Associate an Anycast static IP list with a new distribution
- Associate an Anycast static IP list with a connection group

Step 5: Prepare for migration

For more information, see Step 4: Prepare for migration in the *Amazon VPC User Guide*.

Step 6: Advertise CIDR globally

For more information, see Step 5: Advertise CIDR globally in the *Amazon VPC User Guide*.

Using gRPC with CloudFront distributions

Amazon CloudFront supports gRPC, an open-source remote procedure call (RPC) framework built on HTTP/2. gRPC offers bi-directional streaming and binary protocol that buffers payloads, making it suitable for applications that require low latency communications.

CloudFront receives your gRPC requests and proxies them directly to your origins. You can use CloudFront to proxy four types of gRPC services:

- Unary RPC
- Server streaming RPC
- Client streaming RPC
- Bi-directional streaming RPC

How gRPC works in CloudFront

To configure gRPC in CloudFront, set an origin that provides a gRPC service as your distribution's origin. You can use origins that provide both non-gRPC and gRPC services. CloudFront determines if the incoming request is a gRPC request or an HTTP/HTTPS request based on the `Content-Type` header. If a request's `Content-Type` header has value of `application/grpc`, the request is considered a gRPC request and CloudFront will proxy the request to your origin.

> **Note**
>
> To enable a distribution to handle gRPC requests, include HTTP/2 as one of the supported HTTP versions, and allow HTTP methods, including `POST`. Your gRPC origin endpoint must be configured to support HTTPS, as CloudFront only supports secure (HTTPS-based) gRPC connections. gRPC only supports end-to-end HTTPS. If you're using a custom origin, verify that your Protocol settings support HTTPS.

To enable gRPC support for your distribution, complete the following steps:

1. Update your distribution's cache behavior to allow HTTP methods, including the `POST` method.

2. After you select the POST method, select the gRPC checkbox that appears.
3. Specify **HTTP/2** as one of the supported HTTP versions.

For more information, see the following topics:

- Allow gRPC requests over HTTP/2
- GrpcConfig in the *Amazon CloudFront API Reference*

Because gRPC is used only for non-cacheable API traffic, your cache configurations won't affect gRPC requests. You can use an origin request policy to add custom headers to the gRPC requests that are sent to your gRPC origin. You can use AWS WAF with CloudFront to manage access to your gRPC distribution, control bots, and protect your gRPC applications from web exploits. CloudFront gRPC supports CloudFront Functions.

In addition to HTTP status, you will receive grpc-status along with your gRPC response. For a list of possible values for grpc-status, see Status codes and their use in gRPC.

Notes

gRPC doesn't support the following CloudFront features:

- Custom error responses
- Origin failover isn't supported with gRPC, as gRPC uses POST method. CloudFront fails over to the secondary origin only when the HTTP method of the viewer request is GET, HEAD, or OPTIONS.
- CloudFront proxies gRPC requests directly to the origin and bypasses the Regional Edge Cache (REC). Because gRPC bypasses the REC, gRPC doesn't support Lambda@Edge or Origin Shield.
- gRPC doesn't support AWS WAF request body inspection rules. If you enabled these rules on the web ACL for a distribution, any request that uses gRPC will ignore the request body inspection rules. All other AWS WAF rules will still apply. For more information, see Enable AWS WAF for distributions.

Working with shared resources in CloudFront

Amazon CloudFront integrates with AWS Resource Access Manager (AWS RAM) to enable resource sharing. AWS RAM enables you to share some CloudFront resources with other AWS accounts or through AWS Organizations. With AWS RAM, you share resources that you own by creating a *resource share*. A resource share specifies the resources to share, and the consumers with whom to share them. Consumers can include:

- Specific AWS accounts inside or outside of its organization in AWS Organizations
- An organizational unit inside its organization in AWS Organizations
- Its entire organization in AWS Organizations

For more information about AWS RAM, see the AWS RAM User Guide.

This topic explains how to share resources that you own, and how to use resources that are shared with you.

Contents

- Prerequisites for sharing resources
- Sharing a VPC origin
- Using a shared VPC origin
- Identifying a shared VPC origin
- Unsharing a shared VPC origin
- Responsibilities and permissions for shared VPC origins
- Billing and metering
- Shared resource quotas

Prerequisites for sharing resources

- You must have the AWSRAMDefaultPermissionCloudFront managed policy to grant read-only access to the resource share. For more information, see AWSRAMDefaultPermissionCloudFront.
- To share a VPC origin, you must own it in your AWS account. This means that the resource must be allocated or provisioned in your account. You can't share a resource that has been shared with you.

- To share a resource with your organization or an organizational unit in AWS Organizations, you must enable sharing with AWS Organizations. For more information, see Enable Sharing with AWS Organizations in the *AWS RAM User Guide*.

Sharing a VPC origin

> **Note**
>
> Currently, CloudFront supports sharing VPC origins. If you haven't created one already, see Restrict access with VPC origins.

When you share a VPC origin that you own with other AWS accounts, you enable them to use that resource as the origin for their CloudFront distributions.

To share a VPC origin, you must add it to a resource share. A resource share is an AWS RAM resource that lets you share your resources across AWS accounts.

A resource share specifies the following:

- The resources that you want to share
- The consumers with whom they are shared
- The service's managed policy that determines the permissions to the resources

When you share a VPC origin using the CloudFront console, you add it to an existing resource share. If you don't have a resource share already you can create one when you're sharing a VPC origin from the CloudFront console. You can also use the AWS RAM console or AWS CLI to create one separately.

You can share VPC origins with other AWS accounts and AWS Organizations.

- If you're sharing the resource with an AWS organization, all consumers in that specific organization are allowed access to the VPC origin.
- If you're sharing with an AWS account or an organization that you're not part of, the consumers will receive an invitation to accept the resource share. Once accepted, they can use the VPC origin.

You can share a VPC origin that you own using the CloudFront console, the AWS RAM console, or the AWS CLI.

To create a resource share by using the CloudFront console

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **VPC origins**.
3. Select one or more resources and choose **Share VPC origin**.
4. Choose **Create resource share**.
5. For **Name**, enter a name for the resource share.
6. For **Principal type**, select one of the following options:

   - **AWS account** – Grant access to a specific AWS account.
   - **Organizational unit** – Grant access to a specific organizational unit (OU).
   - **Organization** – Grant access to your entire organization, including its child OUs and AWS accounts.

   a. If you chose **AWS account**, enter the account ID number. You can choose **Add new account** to add up to 5 AWS accounts.

   b. If you chose **Organizational unit**, enter the OU unit ARN. You can enter only 1 OU.

   c. If you chose **Organization**, enter the organization ARN. You can enter only 1 organization.
7. Choose **Share resources**.

   By default, CloudFront applies the AWSRAMDefaultPermissionCloudFront AWS managed policy on the resource share. This policy allows read-only actions on the resource share, so that consuming accounts can't update or delete the shared resource. You can't edit or remove this policy from the resource share.

   Tip

   After you create the resource share, you can add additional AWS accounts from the AWS RAM console. For more information, see Update a resource share in AWS RAM in the *AWS RAM User Guide*.

To share a VPC origin that you own using the CloudFront console

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **VPC origins**.
3. Select a resource and choose **Share VPC origin**.
4. On the **Share VPC origin** page, you can select an existing resource share that you want to add this VPC origin to.
5. Choose **Share resource**.

   On the resource detail page, under **Shared with**, you can see that your VPC origin is shared with the following details:

   - Resource share names
   - Share status
   - Last modified date

After you create and share the resource share with the consuming accounts, they have 12 hours to accept the invitation. For more information, see Accepting and rejecting resource share invitations in the *AWS RAM User Guide*.

> **Important**
>
> To enable consuming accounts to use your VPC origin for their CloudFront distribution, you must also give them the VPC origin's Elastic Load Balancing or Amazon EC2 endpoint.

To share a VPC origin that you own using the AWS RAM console

Create a resource share and then choose the CloudFront resources that you want to add to it. For more information, see Creating a Resource Share in the *AWS RAM User Guide*.

To share a VPC origin that you own using the AWS CLI

Use the create-resource-share command.

Using a shared VPC origin

To use a shared VPC origin, the account that receives the invitation must accept the resource share. You can do that by navigating to the AWS Resource Access Manager console in the US East (N. Virginia) Region and accepting any pending requests in the **Pending** tab. For more information, see Accepting shared resources in the *AWS RAM User Guide*.

After you accept the resource share, you can then use the VPC origin as the origin for your CloudFront distributions.

To use a shared VPC origin

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. On the navigation pane, for **Distributions**, do one of the following:
   - For a new distribution, choose **Create distribution**.
   - For an existing distribution, choose the distribution ID.
3. For the **Origin type**, choose **VPC origin**, and then specify the VPC origin that was shared with you.
4. For **VPC origin endpoint**, enter the private DNS name of your Amazon EC2 instance or Elastic Load Balancing load balancer, or the origin domain. If you don't already have this value, you must get it from the AWS account that owns the VPC origin. If you don't already have this endpoint, you can get it from the AWS account that owns the VPC origin.
5. Follow the rest of the console steps to create or update your distribution.

Identifying a shared VPC origin

Owners and consumers can identify shared VPC origins using the CloudFront console and AWS CLI.

To identify a shared VPC origin using the CloudFront console

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **VPC origins**. You can use the **Owner ID** column to identify the AWS account that the resource belongs to.
3. Select a resource.

4. On the resource detail page, under **Shared with**, you can see that your VPC origin is shared with the following details:

   - Resource share names
   - Share status
   - Last modified date

Unsharing a shared VPC origin

When you unshare a resource, the AWS accounts (consuming accounts) can no longer use that resource for new distributions or update existing distributions.

> **Note**
>
> If you unshare a resource, existing distributions that are still using that resource remain active and will continue to serve traffic. However, these distributions can't be edited until the unshared resource is removed as the origin. We recommend that you make sure that any consuming accounts stop using the unshared resource before you unshare it.

To unshare a shared VPC origin that you own, you must remove it from the resource share. You can do this using the CloudFront console, AWS RAM console, or the AWS CLI.

To unshare a shared VPC origin that you own using the CloudFront console

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **VPC origins**.
3. Select a resource and choose **Unshare**.
4. Review the details in the **Unshare resource** dialog box and then choose **Unshare**. The principals listed will no longer have access to your shared resource.

To unshare a shared VPC origin that you own using the AWS RAM console

See Updating a Resource Share in the *AWS RAM User Guide*.

To unshare a shared VPC origin that you own using the AWS CLI

Use the disassociate-resource-share command.

Responsibilities and permissions for shared VPC origins

Permissions for owners

As the resource-owning account, make sure that any consuming accounts stop using the resource before you unshare or delete it.

Permissions for consumers

Consuming accounts can use shared resources as origins for their CloudFront distributions, but they can't edit or delete the resources. By default, the AWSRAMDefaultPermissionCloudFront AWS managed policy is applied to the resource share in the sharing account (the account that owns the resource).

AWSRAMDefaultPermissionCloudFront

When you create a resource share in CloudFront, CloudFront uses the **AWSRAMDefaultPermissionCloudFront** AWS managed policy and applies it to your resource share. This policy grants read-only permissions to CloudFront resources that can be shared from the resource owner to the consuming account.

For more information about managing permissions in AWS RAM, see Managing permissions in AWS RAM in the *AWS Resource Access Manager User Guide*.

Billing and metering

There are no extra charges for sharing VPC origins with other AWS accounts. The usage costs of traffic for a distribution that is using a shared VPC origin will go to the consuming account that owns the distribution.

Shared resource quotas

CloudFront uses the same resource share quotas as specified by AWS RAM. From the CloudFront console, you can add up to 5 AWS accounts, 1 OU, or 1 organization. To add more, use the AWS RAM console or AWS RAM API.

For more information, see Service quotas for AWS RAM in the *AWS RAM User Guide*.

Caching and availability

You can use CloudFront to reduce the number of requests that your origin server must respond to directly. With CloudFront caching, more objects are served from CloudFront edge locations, which are closer to your users. This reduces the load on your origin server and reduces latency.

The more requests that CloudFront can serve from edge caches, the fewer viewer requests that CloudFront must forward to your origin to get the latest version or a unique version of an object. To optimize CloudFront to make as few requests to your origin as possible, consider using a CloudFront Origin Shield. For more information, see Use Amazon CloudFront Origin Shield.

The proportion of requests that are served directly from the CloudFront cache compared to all requests is called the *cache hit ratio*. You can view the percentage of viewer requests that are hits, misses, and errors in the CloudFront console. For more information, see View CloudFront cache statistics reports.

A number of factors affect the cache hit ratio. You can adjust your CloudFront distribution configuration to improve the cache hit ratio by following the guidance in Increase the proportion of requests that are served directly from the CloudFront caches (cache hit ratio).

To learn about adding and removing the content that you want CloudFront to serve, see Add, remove, or replace content that CloudFront distributes.

Topics

- Increase the proportion of requests that are served directly from the CloudFront caches (cache hit ratio)
- Use Amazon CloudFront Origin Shield
- Optimize high availability with CloudFront origin failover
- Manage how long content stays in the cache (expiration)
- Cache content based on query string parameters
- Cache content based on cookies
- Cache content based on request headers

Increase the proportion of requests that are served directly from the CloudFront caches (cache hit ratio)

You can improve performance by increasing the proportion of your viewer requests that are served directly from the CloudFront cache instead of going to your origin servers for content. This is known as improving the cache hit ratio.

The following sections explain how to improve your cache hit ratio.

Topics

- Specify how long CloudFront caches your objects
- Use Origin Shield
- Caching based on query string parameters
- Caching based on cookie values
- Caching based on request headers
- Remove Accept-Encoding header when compression is not needed
- Serve media content over HTTP

Specify how long CloudFront caches your objects

To increase your cache hit ratio, you can configure your origin to add a Cache-Control max-age directive to your objects, and specify the longest practical value for `max-age`. The shorter the cache duration, the more frequently CloudFront sends requests to your origin to determine if an object has changed and to get the latest version. You can supplement `max-age` with the `stale-while-revalidate` and `stale-if-error` directives to further improve cache hit ratio under certain conditions. For more information, see Manage how long content stays in the cache (expiration).

Use Origin Shield

CloudFront Origin Shield can help improve the cache hit ratio of your CloudFront distribution, because it provides an additional layer of caching in front of your origin. When you use Origin Shield, all requests from all of the CloudFront caching layers to your origin come from a single location. CloudFront can retrieve each object using a single origin request from Origin Shield, and

all other layers of the CloudFront cache (edge locations and regional edge caches) can retrieve the object from Origin Shield.

For more information, see Use Amazon CloudFront Origin Shield.

Caching based on query string parameters

If you configure CloudFront to cache based on query string parameters, you can improve caching if you do the following:

- Configure CloudFront to forward only the query string parameters for which your origin will return unique objects.
- Use the same case (uppercase or lowercase) for all instances of the same parameter. For example, if one request contains `parameter1=A` and another contains `parameter1=a`, CloudFront forwards separate requests to your origin when a request contains `parameter1=A` and when a request contains `parameter1=a`. CloudFront then separately caches the corresponding objects returned by your origin even if the objects are identical. If you use just `A` or `a`, CloudFront forwards fewer requests to your origin.
- List parameters in the same order. As with differences in case, if one request for an object contains the query string `parameter1=a&parameter2=b` and another request for the same object contains `parameter2=b&parameter1=a`, CloudFront forwards both requests to your origin and separately caches the corresponding objects even if they're identical. If you always use the same order for parameters, CloudFront forwards fewer requests to your origin.

For more information, see Cache content based on query string parameters. If you want to review the query strings that CloudFront forwards to your origin, see the values in the `cs-uri-query` column of your CloudFront log files. For more information, see Access logs (standard logs).

Caching based on cookie values

If you configure CloudFront to cache based on cookie values, you can improve caching if you do the following:

- Configure CloudFront to forward only specified cookies instead of forwarding all cookies. For the cookies that you configure CloudFront to forward to your origin, CloudFront forwards every combination of cookie name and value. It then separately caches the objects that your origin returns, even if they're all identical.

For example, suppose that viewers include two cookies in every request, that each cookie has three possible values, and that all combinations of cookie values are possible. CloudFront forwards up to nine different requests to your origin for each object. If your origin returns different versions of an object based on only one of the cookies, then CloudFront is forwarding more requests to your origin than necessary and is needlessly caching multiple identical versions of the object.

- Create separate cache behaviors for static and dynamic content, and configure CloudFront to forward cookies to your origin only for dynamic content.

  For example, suppose you have just one cache behavior for your distribution and that you're using the distribution both for dynamic content, such as `.js` files, and for `.css` files that rarely change. CloudFront caches separate versions of your `.css` files based on cookie values, so each CloudFront edge location forwards a request to your origin for every new cookie value or combination of cookie values.

  Suppose you create a cache behavior whose path pattern is `*.css`, and the behavior doesn't cache based on cookie values. In that case, CloudFront forwards requests for `.css` files to your origin for only the first request that an edge location receives for a given `.css` file and for the first request after a `.css` file expires.
- If possible, create separate cache behaviors for dynamic content when cookie values are unique for each user (such as a user ID), and dynamic content that varies based on a smaller number of unique values.

For more information, see Cache content based on cookies. If you want to review the cookies that CloudFront forwards to your origin, see the values in the `cs(Cookie)` column of your CloudFront log files. For more information, see Access logs (standard logs).

Caching based on request headers

If you configure CloudFront to cache based on request headers, you can improve caching if you do the following:

- Configure CloudFront to forward and cache based on only specified headers instead of forwarding and caching based on all headers. For the headers that you specify, CloudFront forwards every combination of header name and value. It then separately caches the objects that your origin returns even if they're all identical.

> **Note**
>
> CloudFront always forwards to your origin the headers specified in the following topics:
>
> - How CloudFront processes and forwards requests to your Amazon S3 origin server > HTTP request headers that CloudFront removes or updates
> - How CloudFront processes and forwards requests to your custom origin server > HTTP request headers and CloudFront behavior (custom and Amazon S3 origins)

When you configure CloudFront to cache based on request headers, you don't change the headers that CloudFront forwards, only whether CloudFront caches objects based on the header values.

- Try to avoid caching based on request headers that have large numbers of unique values.

  For example, if you want to serve different sizes of an image based on the user's device, then don't configure CloudFront to cache based on the `User-Agent` header, which has an enormous number of possible values. Instead, configure CloudFront to cache based on the CloudFront device-type headers `CloudFront-Is-Desktop-Viewer`, `CloudFront-Is-Mobile-Viewer`, `CloudFront-Is-SmartTV-Viewer`, and `CloudFront-Is-Tablet-Viewer`. In addition, if you're returning the same version of the image for tablets and desktops, then forward only the `CloudFront-Is-Tablet-Viewer` header, not the `CloudFront-Is-Desktop-Viewer` header.

For more information, see Cache content based on request headers.

Remove `Accept-Encoding` header when compression is not needed

If compression is not enabled—because the origin doesn't support it, CloudFront doesn't support it, or the content is not compressible—you can increase the cache hit ratio by associating a cache behavior in your distribution to an origin that sets the Custom Origin Header as follows:

- **Header name**: `Accept-Encoding`
- **Header value**: (Keep blank)

When you use this configuration, CloudFront removes the `Accept-Encoding` header from the cache key and doesn't include the header in origin requests. This configuration applies to all content that CloudFront serves with the distribution from that origin.

Serve media content over HTTP

For information about optimizing video on demand (VOD) and streaming video content, see Video on demand and live streaming video with CloudFront.

Use Amazon CloudFront Origin Shield

CloudFront Origin Shield is an additional layer in the CloudFront caching infrastructure that helps to minimize your origin's load, improve its availability, and reduce its operating costs. With CloudFront Origin Shield, you get the following benefits:

Better cache hit ratio

Origin Shield can help improve the cache hit ratio of your CloudFront distribution because it provides an additional layer of caching in front of your origin. When you use Origin Shield, all requests from all of the CloudFront caching layers to your origin go through Origin Shield, increasing the likelihood of a cache hit. Requests that are made in the same Region as your origin will bypass Origin Shield. CloudFront can retrieve each object with a single origin request from Origin Shield to your origin, and all other layers of the CloudFront cache (edge locations and regional edge caches) can retrieve the object from Origin Shield.

Reduced origin load

Origin Shield can further reduce the number of simultaneous requests that are sent to your origin for the same object. Requests for content that is not in Origin Shield's cache are consolidated with other requests for the same object, resulting in as few as one request going to your origin. Handling fewer requests at your origin can preserve your origin's availability during peak loads or unexpected traffic spikes, and can reduce costs for things like just-in-time packaging, image transformations, and data transfer out (DTO).

Better network performance

When you enable Origin Shield in the AWS Region that has the lowest latency to your origin, you can get better network performance. For origins in an AWS Region, CloudFront network traffic remains on the high throughput CloudFront network all the way to your origin. For origins outside of AWS, CloudFront network traffic remains on the CloudFront network all the way to Origin Shield, which has a low latency connection to your origin.

You incur additional charges for using Origin Shield. For more information, see CloudFront pricing.

> **Note**
>
> Origin Shield isn't supported with gRPC requests. If a distribution that supports gRPC has Origin Shield enabled, the gRPC requests will continue to work. However, the requests will be proxied directly to the gRPC origin without going through Origin Shield. For more information, see Using gRPC with CloudFront distributions.

Topics

- Use cases for Origin Shield
- Choose the AWS Region for Origin Shield
- Enable Origin Shield
- Estimate Origin Shield costs
- Origin Shield high availability
- How Origin Shield interacts with other CloudFront features

Use cases for Origin Shield

CloudFront Origin Shield can be beneficial for many use cases, including the following:

- Viewers that are spread across different geographical regions
- Origins that provide just-in-time packaging for live streaming or on-the-fly image processing
- On-premises origins with capacity or bandwidth constraints
- Workloads that use multiple content delivery networks (CDNs)

Origin Shield may not be a good fit in other cases, such as dynamic content that is proxied to the origin, content with low cacheability, or content that is infrequently requested.

The following sections explain the benefits of Origin Shield for the following use cases.

Use cases

- Viewers in different geographical regions
- Multiple CDNs

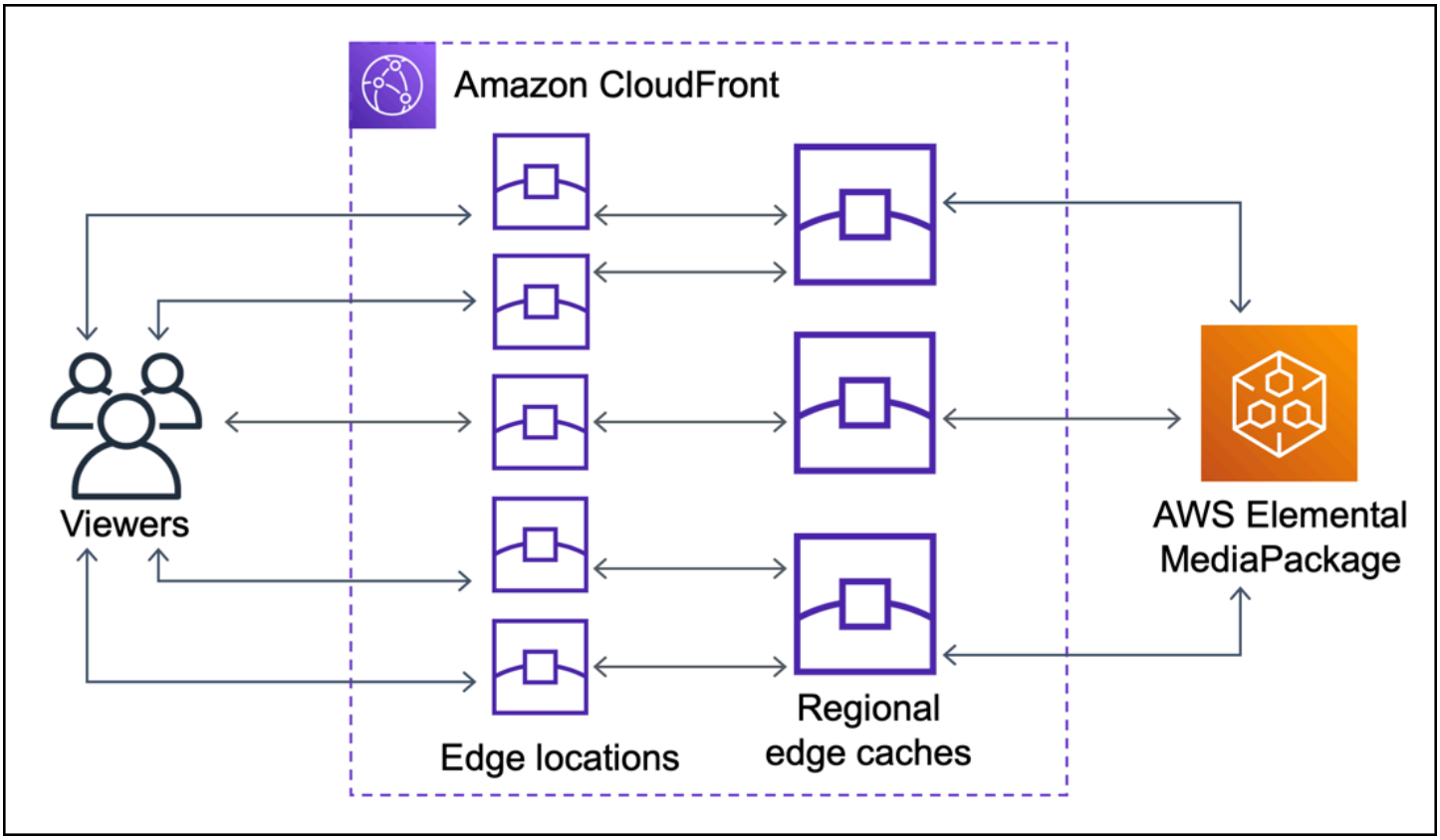
Viewers in different geographical regions

With Amazon CloudFront, you inherently get a reduced load on your origin because requests that CloudFront can serve from the cache don't go to your origin. In addition to CloudFront global network of edge locations, regional edge caches serve as a mid-tier caching layer to provide cache hits and consolidate origin requests for viewers in nearby geographical regions. Viewer requests are routed first to a nearby CloudFront edge location, and if the object isn't cached in that location, the request is sent on to a regional edge cache.

When viewers are in different geographical regions, requests can be routed through different regional edge caches, each of which can send a request to your origin for the same content. But with Origin Shield, you get an additional layer of caching between the regional edge caches and your origin. All requests from all regional edge caches go through Origin Shield, further reducing the load on your origin. The following diagrams illustrate this. In the following diagrams, the origin is AWS Elemental MediaPackage.

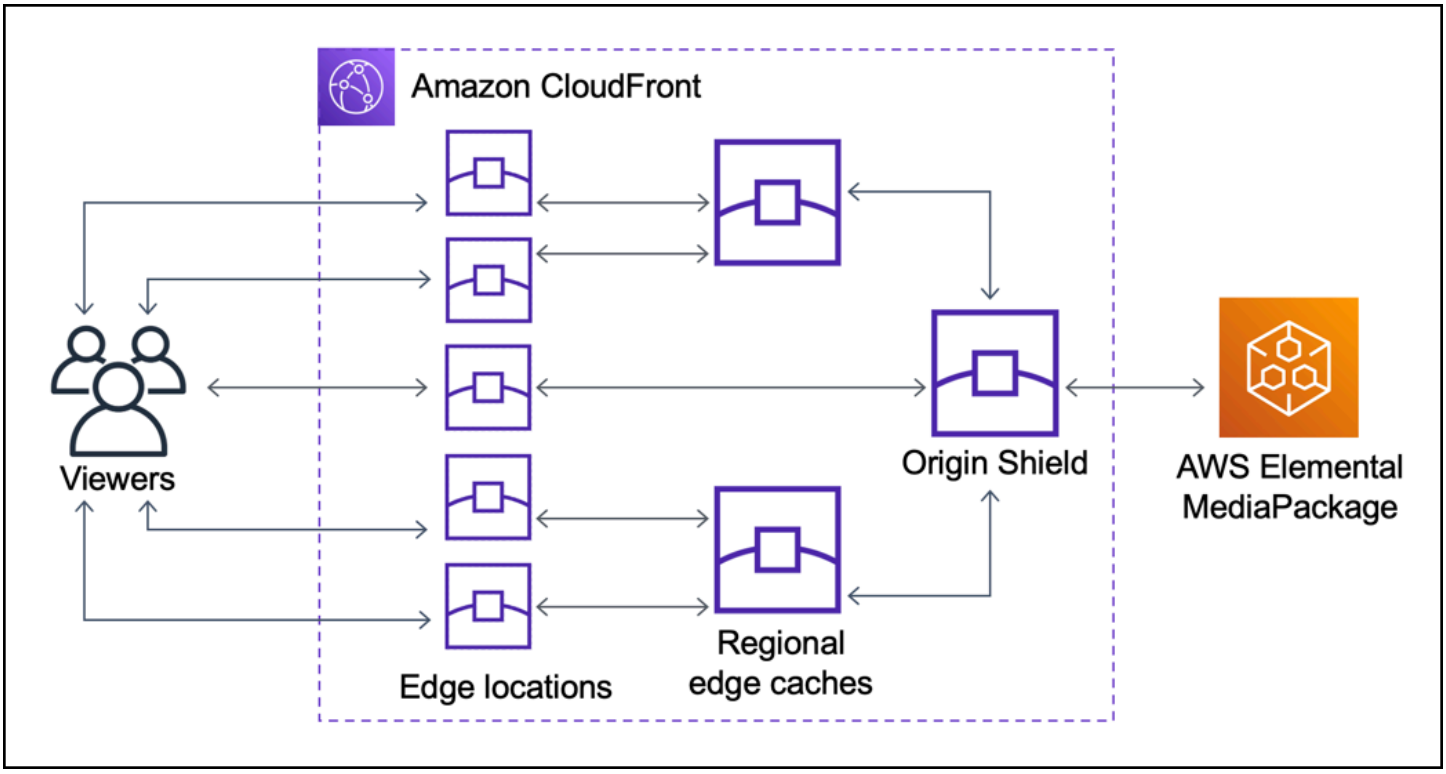
Without Origin Shield

Without Origin Shield, your origin might receive duplicate requests for the same content, as shown in the following diagram.



With Origin Shield

Using Origin Shield can help reduce the load on your origin, as shown in the following diagram.



Multiple CDNs

To serve live video events or popular on-demand content, you might use multiple content delivery networks (CDNs). Using multiple CDNs can offer certain advantages, but it also means that your origin might receive many duplicate requests for the same content, each coming from different CDNs or different locations within the same CDN. These redundant requests might adversely affect the availability of your origin or cause additional operating costs for processes like just-in-time packaging or data transfer out (DTO) to the internet.

When you combine Origin Shield with using your CloudFront distribution as the origin for other CDNs, you can get the following benefits:

- Fewer redundant requests received at your origin, which helps to reduce the negative effects of using multiple CDNs.
- A common cache key across CDNs, and centralized management for origin-facing features.
- Improved network performance. Network traffic from other CDNs is terminated at a nearby CloudFront edge location, which might provide a hit from the local cache. If the requested object

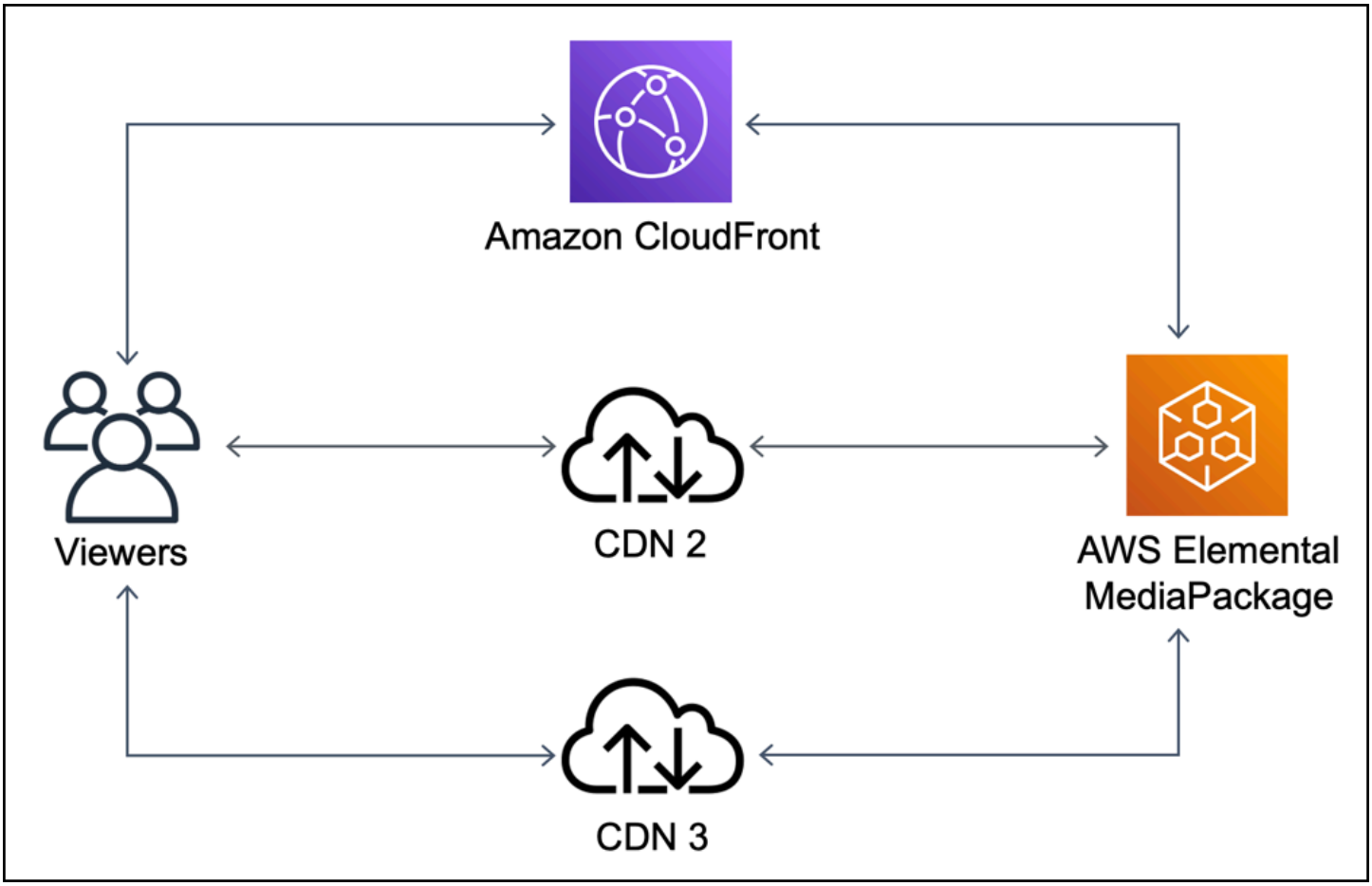
is not in the edge location cache, the request to the origin remains on the CloudFront network all the way to Origin Shield, which provides high throughput and low latency to the origin. If the requested object is in Origin Shield's cache, the request to your origin is avoided entirely.

> **Important**
>
> If you are interested in using Origin Shield in a multi-CDN architecture, and have discounted pricing, contact us or your AWS sales representative for more information. Additional charges may apply.

The following diagrams show how this configuration can help minimize the load on your origin when you serve popular live video events with multiple CDNs. In the following diagrams, the origin is AWS Elemental MediaPackage.

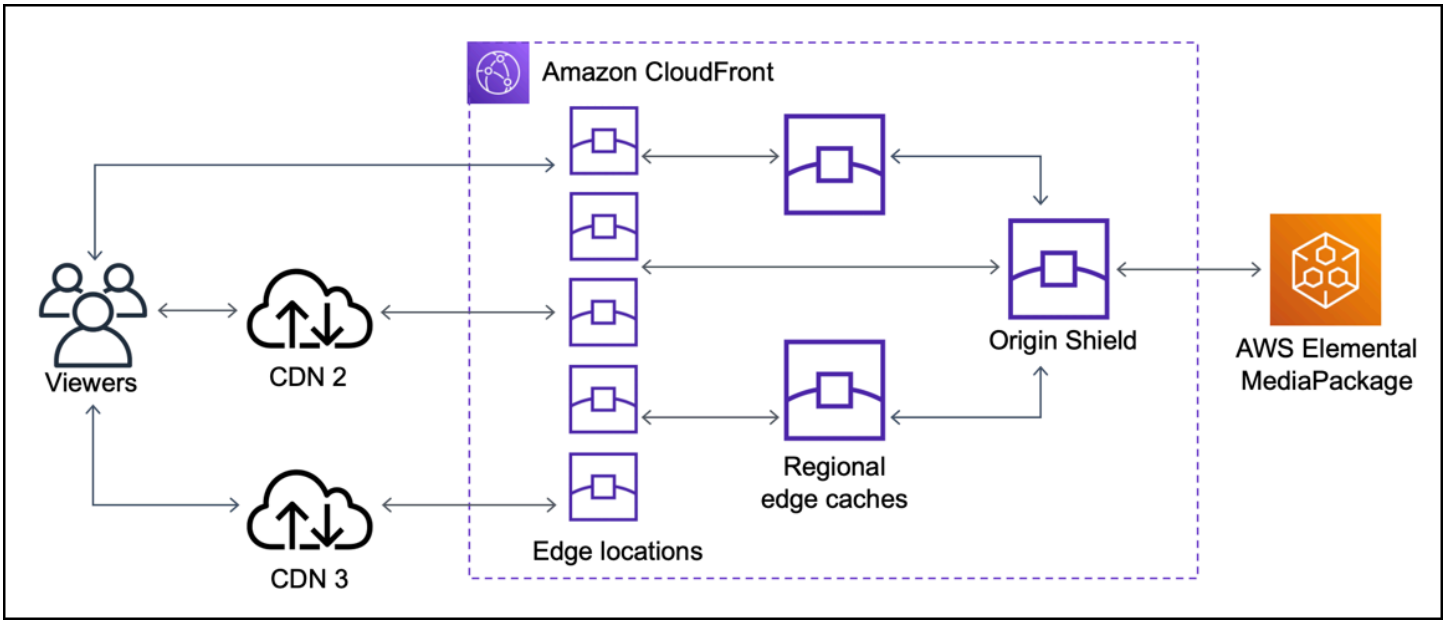
Without Origin Shield (multiple CDNs)

Without Origin Shield, your origin might receive many duplicate requests for the same content, each coming from a different CDN, as shown in the following diagram.



With Origin Shield (multiple CDNs)

Using Origin Shield, with CloudFront as the origin for your other CDNs, can help reduce the load on your origin, as shown in the following diagram.



Choose the AWS Region for Origin Shield

Amazon CloudFront offers Origin Shield in AWS Regions where CloudFront has a regional edge cache. When you enable Origin Shield, you choose the AWS Region for Origin Shield. You should choose the AWS Region that has the lowest latency to your origin. You can use Origin Shield with origins that are in an AWS Region, and with origins that are not in AWS.

For origins in an AWS Region

If your origin is in an AWS Region, first determine whether your origin is in a Region in which CloudFront offers Origin Shield. CloudFront offers Origin Shield in the following AWS Regions.

- US East (Ohio) – `us-east-2`
- US East (N. Virginia) – `us-east-1`
- US West (Oregon) – `us-west-2`
- Asia Pacific (Mumbai) – `ap-south-1`
- Asia Pacific (Seoul) – `ap-northeast-2`
- Asia Pacific (Singapore) – `ap-southeast-1`
- Asia Pacific (Sydney) – `ap-southeast-2`
- Asia Pacific (Tokyo) – `ap-northeast-1`
- Europe (Frankfurt) – `eu-central-1`

- Europe (Ireland) – `eu-west-1`
- Europe (London) – `eu-west-2`
- Europe (Paris) – `eu-west-3`
- South America (São Paulo) – `sa-east-1`
- Middle East (UAE) – `me-central-1`

If your origin is in an AWS Region in which CloudFront offers Origin Shield

If your origin is in an AWS Region in which CloudFront offers Origin Shield (see the preceding list), enable Origin Shield in the same Region as your origin.

If your origin is not in an AWS Region in which CloudFront offers Origin Shield

If your origin is not in an AWS Region in which CloudFront offers Origin Shield, see the following table to determine which Region to enable Origin Shield in.

If your origin is in ...	Enable Origin Shield in ...
US West (N. California) – `us-west-1`	US West (Oregon) – `us-west-2`
Africa (Cape Town) – `af-south-1`	Europe (Ireland) – `eu-west-1`
Asia Pacific (Hong Kong) – `ap-east-1`	Asia Pacific (Singapore) – `ap-southeast-1`
Canada (Central) – `ca-central-1`	US East (N. Virginia) – `us-east-1`
Europe (Milan) – `eu-south-1`	Europe (Frankfurt) – `eu-central-1`
Europe (Stockholm) – `eu-north-1`	Europe (London) – `eu-west-2`
Middle East (Bahrain) – `me-south-1`	Asia Pacific (Mumbai) – `ap-south-1`

For origins outside of AWS

You can use Origin Shield with an origin that is on-premises or is not in an AWS Region. In this case, enable Origin Shield in the AWS Region that has the lowest latency to your origin. If you're not sure which AWS Region has the lowest latency to your origin, you can use the following suggestions to help you make a determination.

- You can consult the preceding table for an approximation of which AWS Region might have the lowest latency to your origin, based on your origin's geographic location.
- You can launch Amazon EC2 instances in a few different AWS Regions that are geographically close to your origin, and run some tests using `ping` to measure the typical network latencies between those Regions and your origin.

Enable Origin Shield

You can enable Origin Shield to improve your cache hit ratio, reduce the load on your origin, and help improve performance. To enable Origin Shield, change the origin settings in a CloudFront distribution. Origin Shield is a property of the origin. For each origin in your CloudFront distributions, you can separately enable Origin Shield in whichever AWS Region provides the best performance for that origin.

You can enable Origin Shield in the CloudFront console, with CloudFormation, or with the CloudFront API.

Console

To enable Origin Shield for an existing origin (console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the distribution that has the origin that you want to update.
3. Choose the **Origins** tab.
4. Choose the origin to update, then choose **Edit**.
5. For **Enable Origin Shield**, choose **Yes**.
6. For **Origin Shield Region**, choose the AWS Region where you want to enable Origin Shield. For help choosing a Region, see Choose the AWS Region for Origin Shield.
7. Choose **Save changes**.

When your distribution status is **Deployed**, Origin Shield is ready. This takes a few minutes.

To enable Origin Shield for a new origin (console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. To create the new origin in an existing distribution, do the following:

   1. Choose the distribution where you want to create the origin.
   2. Choose **Create Origin**, and then proceed to step 3.

   To create the new origin in a new standard distribution, do the following:

   1. Follow the steps to create a standard distribution in the console. For more information, see Create a CloudFront distribution in the console.
   2. In the **Settings** section, choose **Customize origin settings**. Proceed to step 3.
3. For **Enable Origin Shield**, choose **Yes**.
4. For **Origin Shield Region**, choose the AWS Region where you want to enable Origin Shield. For help choosing a Region, see Choose the AWS Region for Origin Shield.
5. Follow the steps in the console to finish creating your origin or distribution.

When your distribution status is **Deployed**, Origin Shield is ready. This takes a few minutes.

CloudFormation

To enable Origin Shield with CloudFormation, use the `OriginShield` property in the `Origin` property type in an `AWS::CloudFront::Distribution` resource. You can add the `OriginShield` property to an existing `Origin`, or include it when you create a new `Origin`.

The following example shows the syntax, in YAML format, for enabling `OriginShield` in the US West (Oregon) Region (`us-west-2`). For help choosing a Region, see the section called "Choose the AWS Region for Origin Shield". This example shows only the `Origin` property type, not the entire `AWS::CloudFront::Distribution` resource.

```
Origins:
- DomainName: 3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com
  Id: Example-EMP-3ae97e9482b0d011
  OriginShield:
    Enabled: true
    OriginShieldRegion: us-west-2
  CustomOriginConfig:
    OriginProtocolPolicy: match-viewer
    OriginSSLProtocols: TLSv1
```

For more information, see AWS::CloudFront::Distribution Origin in the resource and property reference section of the *AWS CloudFormation User Guide*.

API

To enable Origin Shield with the CloudFront API using the AWS SDKs or AWS Command Line Interface (AWS CLI), use the `OriginShield` type. You specify `OriginShield` in an `Origin`, in a `DistributionConfig`. For information about the `OriginShield` type, see the following information in the *Amazon CloudFront API Reference*.

- OriginShield (type)
- Origin (type)
- DistributionConfig (type)
- UpdateDistribution (operation)
- CreateDistribution (operation)

The specific syntax for using these types and operations varies based on the SDK, CLI, or API client. For more information, see the reference documentation for your SDK, CLI, or client.

Estimate Origin Shield costs

You accrue charges for Origin Shield based on the number of requests that go to Origin Shield as an incremental layer.

For dynamic (non-cacheable) requests that are proxied to the origin, Origin Shield is always an incremental layer. Dynamic requests use the HTTP methods `PUT`, `POST`, `PATCH`, and `DELETE`.

`GET` and `HEAD` requests that have a time to live (TTL) setting of less than 3600 seconds are considered dynamic requests. In addition, `GET` and `HEAD` requests that have disabled caching are also considered dynamic requests.

To estimate your charges for Origin Shield for dynamic requests, use the following formula:

Total number of dynamic requests **x** Origin Shield charge per 10,000 requests **/** 10,000

For non-dynamic requests with the HTTP methods `GET`, `HEAD`, and `OPTIONS`, Origin Shield is sometimes an incremental layer. When you enable Origin Shield, you choose the AWS Region for Origin Shield. For requests that naturally go to the regional edge cache in the same Region

as Origin Shield, Origin Shield will be skipped. You don't accrue Origin Shield charges for these requests. For requests that go to a regional edge cache in a different Region from Origin Shield, and then go to Origin Shield, Origin Shield is an incremental layer. You do accrue Origin Shield charges for these requests.

To estimate your charges for Origin Shield for cacheable requests, use the following formula:

Total number of cacheable requests **x** (1 – cache hit rate) **x** percentage of requests that go to Origin Shield from a regional edge cache in a different region **x** Origin Shield charge per 10,000 requests **/** 10,000

For more information about the charge per 10,000 requests for Origin Shield, see CloudFront Pricing.

Origin Shield high availability

Origin Shield leverages the CloudFront regional edge caches feature. Each of these edge caches is built in an AWS Region using at least three Availability Zones with fleets of auto-scaling Amazon EC2 instances. Connections from CloudFront locations to Origin Shield also use active error tracking for each request to automatically route the request to a secondary Origin Shield location if the primary Origin Shield location is unavailable.

How Origin Shield interacts with other CloudFront features

The following sections explain how Origin Shield interacts with other CloudFront features.

Origin Shield and CloudFront logging

To see when Origin Shield handled a request, you must enable one of the following:

- CloudFront standard logs (access logs). Standard logs are provided free of charge.
- CloudFront real-time access logs. You incur additional charges for using real-time access logs. See Amazon CloudFront Pricing.

Cache hits from Origin Shield appear as `OriginShieldHit` in the `x-edge-detailed-result-type` field in CloudFront logs. Origin Shield leverages Amazon CloudFront regional edge caches. If a request is routed from a CloudFront edge location to the regional edge cache that is acting as Origin Shield, it is reported as a `Hit` in the logs, not as an `OriginShieldHit`.

Origin Shield and origin groups

Origin Shield is compatible with CloudFront origin groups. Because Origin Shield is a property of the origin, requests always travel through Origin Shield for each origin even when the origin is part of an origin group. For a given request, CloudFront routes the request to the primary origin in the origin group through the primary origin's Origin Shield. If that request fails (according to the origin group failover criteria), CloudFront routes the request to the secondary origin through the secondary origin's Origin Shield.

Origin Shield and Lambda@Edge

Origin Shield does not impact the functionality of Lambda@Edge functions, but it can affect the AWS Region where those functions run.

When you use Origin Shield with Lambda@Edge, origin-facing triggers (origin request and origin response) run in the AWS Region where Origin Shield is enabled. If the primary Origin Shield location is unavailable and CloudFront routes requests to a secondary Origin Shield location, Lambda@Edge origin-facing triggers will also shift to use the secondary Origin Shield location.

Viewer-facing triggers are not affected.

Optimize high availability with CloudFront origin failover

You can set up CloudFront with origin failover for scenarios that require high availability. To get started, you create an *origin group* with two origins: a primary and a secondary. If the primary origin is unavailable, or returns specific HTTP response status codes that indicate a failure, CloudFront automatically switches to the secondary origin.

To set up origin failover, you must have a distribution with at least two origins. Next, you create an origin group for your distribution that includes two origins, setting one as the primary. Finally, you create or update a cache behavior to use the origin group.

To see the steps for setting up origin groups and configuring specific origin failover options, see Create an origin group.

After you configure origin failover for a cache behavior, CloudFront does the following for viewer requests:

- When there's a cache hit, CloudFront returns the requested object.

- When there's a cache miss, CloudFront routes the request to the primary origin in the origin group.
- When the primary origin returns a status code that is not configured for failover, such as an HTTP 2xx or 3xx status code, CloudFront serves the requested object to the viewer.
- When any of the following occur:
  - The primary origin returns an HTTP status code that you've configured for failover
  - CloudFront fails to connect to the primary origin (when 503 is set as a failover code)
  - The response from the primary origin takes too long (times out) (when 504 is set as a failover code)

  Then CloudFront routes the request to the secondary origin in the origin group.

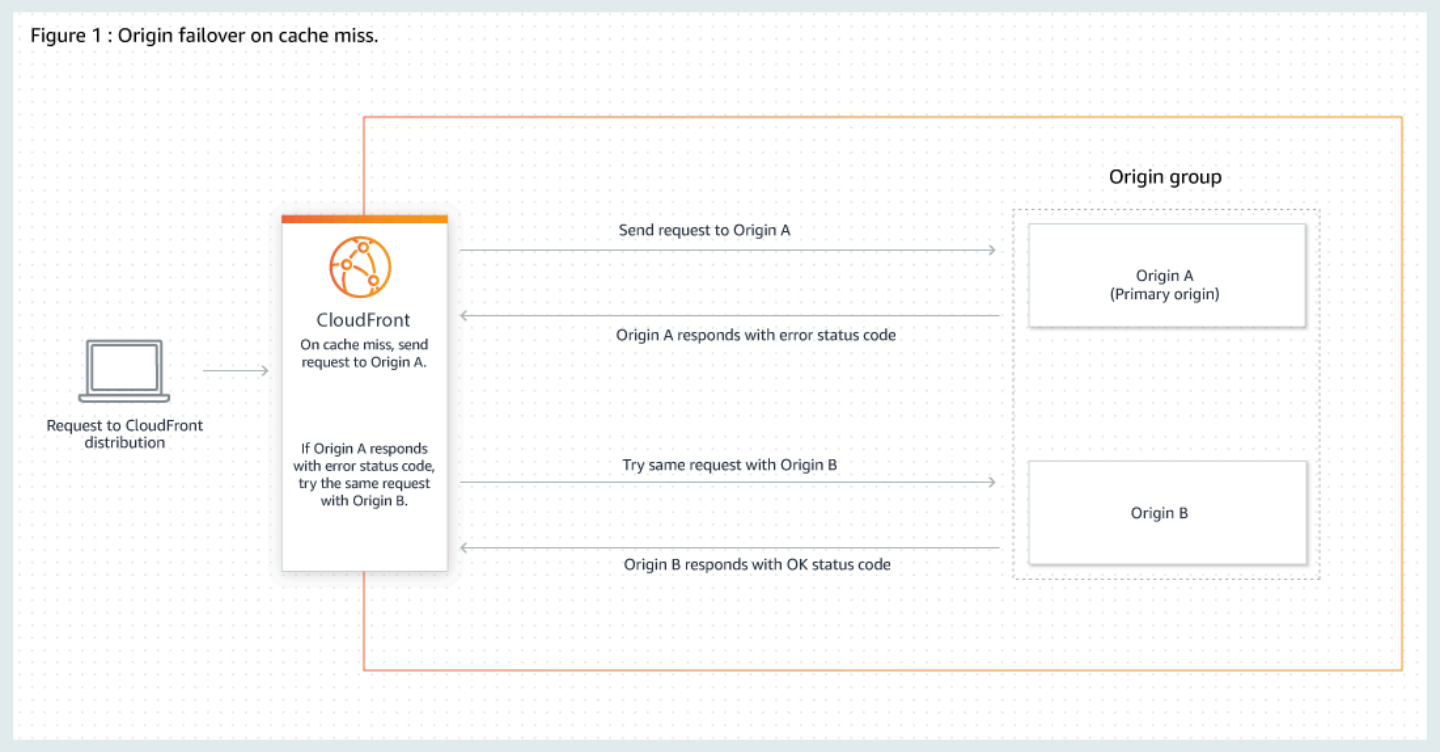
> **Note**
>
> For some use cases, like streaming video content, you might want CloudFront to fail over to the secondary origin quickly. To adjust how quickly CloudFront fails over to the secondary origin, see Control origin timeouts and attempts.

CloudFront routes all incoming requests to the primary origin, even when a previous request failed over to the secondary origin. CloudFront only sends requests to the secondary origin after a request to the primary origin fails.

CloudFront fails over to the secondary origin only when the HTTP method of the viewer request is `GET`, `HEAD`, or `OPTIONS`. CloudFront does not fail over when the viewer sends a different HTTP method (for example `POST`, `PUT`, and so on).

> **Note**
>
> CloudFront will not failover if `OPTIONS` are not set as a Cached HTTP methods in your cache behavior.

The following diagram illustrates how origin failover works.



Topics

- Create an origin group
- Control origin timeouts and attempts
- Use origin failover with Lambda@Edge functions
- Use custom error pages with origin failover

Create an origin group

To create an origin group

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the distribution that you want to create the origin group for.
3. Choose the **Origins** tab.
4. Make sure the distribution has more than one origin. If it doesn't, add a second origin.
5. On the **Origins** tab, in the **Origin groups** pane, choose **Create origin group**.
6. Choose the origins for the origin group. After you add origins, use the arrows to set the priority —that is, which origin is primary and which is secondary.

7. Enter a name for the origin group.
8. Choose the HTTP status codes to use as failover criteria. You can choose any combination of the following status codes: 400, 403, 404, 416, 429, 500, 502, 503, or 504. When CloudFront receives a response with one of the status codes that you specify, it fails over to the secondary origin.

   > **Note**
   >
   > CloudFront fails over to the secondary origin only when the HTTP method of the viewer request is `GET`, `HEAD`, or `OPTIONS`. CloudFront does not fail over when the viewer sends a different HTTP method (for example `POST`, `PUT`, and so on).

9. Under **Origin selection criteria**, specify how your origins are selected when your distribution routes viewer requests. You can choose the following options.

   Default

   CloudFront will use the default origin priority that you specify on the **Settings** page.

   Media quality score

   CloudFront tracks and uses this score to determine the first origin to forward the request to. This also authorizes CloudFront to make asynchronous `HEAD` requests to the alternate origin in the origin group to determine its media quality score. You can only choose this option for AWS Elemental MediaPackage v2 origins. For more information, see Media quality-aware resiliency.

10. Choose **Create origin group**.

Make sure to assign your origin group as the origin for your distribution's cache behavior. For more information, see Name.

Control origin timeouts and attempts

By default, CloudFront tries to connect to the primary origin in an origin group for as long as 30 seconds (3 connection attempts of 10 seconds each) before failing over to the secondary origin. For some use cases, like streaming video content, you might want CloudFront to fail over to the secondary origin more quickly. You can adjust the following settings to affect how quickly CloudFront fails over to the secondary origin. If the origin is a secondary origin, or an origin that

is not part of an origin group, these settings affect how quickly CloudFront returns an HTTP 504 response to the viewer.

To fail over more quickly, specify a shorter connection timeout, fewer connection attempts, or both. For custom origins (including Amazon S3 bucket origins that *are* configured with static website hosting), you can also adjust the origin response timeout.

Origin connection timeout

The origin connection timeout setting affects how long CloudFront waits when trying to establish a connection to the origin. By default, CloudFront waits 10 seconds to establish a connection, but you can specify 1–10 seconds (inclusive). For more information, see Connection timeout.

Origin connection attempts

The origin connection attempts setting affects the number of times that CloudFront attempts to connect to the origin. By default, CloudFront tries 3 times to connect, but you can specify 1–3 (inclusive). For more information, see Connection attempts.

For a custom origin (including an Amazon S3 bucket that's configured with static website hosting), this setting also affects the number of times that CloudFront attempts to get a response from the origin in the case of an origin response timeout.

Origin response timeout

The origin response timeout, also known as the origin read timeout, affects how long CloudFront waits to receive a response (or to receive the complete response) from the origin. By default, CloudFront waits for 30 seconds, but you can specify 1–120 seconds (inclusive). For more information, see Response timeout.

How to change these settings

To change these settings in the CloudFront console

- For a new origin or a new distribution, you specify these values when you create the resource.
- For an existing origin in an existing distribution, you specify these values when you edit the origin.

For more information, see All distribution settings reference.

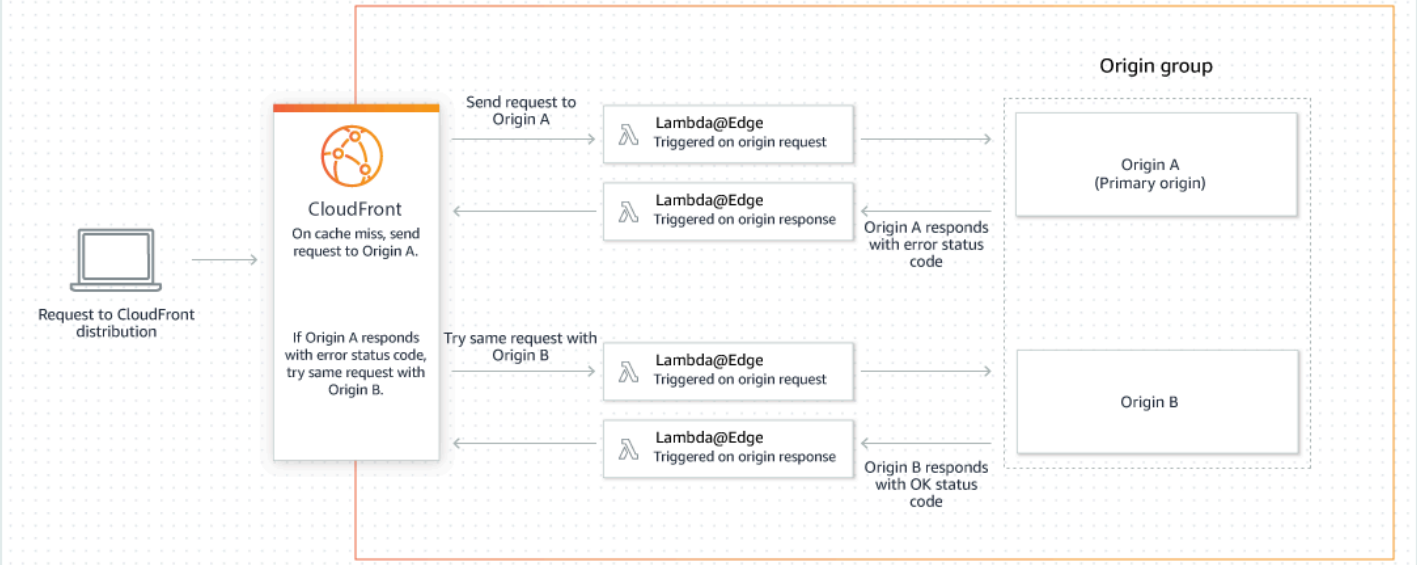
Use origin failover with Lambda@Edge functions

You can use Lambda@Edge functions with CloudFront distributions that you've set up with origin groups. To use a Lambda function, specify it in an origin request or origin response trigger for an origin group when you create the cache behavior. When you use a Lambda@Edge function with an origin group, the function can be triggered twice for a single viewer request. For example, consider this scenario:

1. You create a Lambda@Edge function with an origin request trigger.
2. The Lambda function is triggered once when CloudFront sends a request to the primary origin (on a cache miss).
3. The primary origin responds with an HTTP status code that's configured for failover.
4. The Lambda function is triggered again when CloudFront sends the same request to the secondary origin.

The following diagram illustrates how origin failover works when you include a Lambda@Edge function in an origin request or response trigger.

Figure 2 : Origin failover with Lambda@Edge functions triggered on origin request and response events.



For more information about using Lambda@Edge triggers, see the section called "Add triggers for a Lambda@Edge function".

For more information about managing DNS failover, see Configuring DNS failover in the *Amazon Route 53 Developer Guide*.

Use custom error pages with origin failover

You can use custom error pages with origin groups similarly to how you use them with origins that are not set up for origin failover.

When you use origin failover, you can configure CloudFront to return a custom error page for the primary or secondary origin (or both):

- **Return a custom error page for the primary origin** – If the primary origin returns an HTTP status code that's not configured for failover, CloudFront returns the custom error page to viewers.
- **Return a custom error page for the secondary origin** – If CloudFront receives a failure status code from the secondary origin, CloudFront returns the custom error page.

For more information about using custom error pages with CloudFront, see Generate custom error responses.

Manage how long content stays in the cache (expiration)

You can control how long your files stay in a CloudFront cache before CloudFront forwards another request to your origin. By reducing the duration, you can serve dynamic content. Increasing the duration means that your users get better performance because your files are more likely to be served directly from the edge cache. A longer duration also reduces the load on your origin.

Typically, CloudFront serves a file from an edge location until the cache duration that you specified passes—that is, until the file expires. After it expires, the next time the edge location gets a request for the file, CloudFront forwards the request to the origin to verify that the cache contains the latest version of the file. The response from the origin depends on whether the file has changed:

- If the CloudFront cache already has the latest version, the origin returns a status code `304 Not Modified`.
- If the CloudFront cache does not have the latest version, the origin returns a status code `200 OK` and the latest version of the file.

If a file in an edge location isn't frequently requested, CloudFront might evict the file—remove the file before its expiration date—to make room for files that have been requested more recently.

We recommend managing your cache duration by updating your distribution's cache policy. If you opt out of using a cache policy, the default TTL (Time to Live) is 24 hours, but you can update the following settings to override the default:

- To change the cache duration for all files that match the same path pattern, you can change the CloudFront settings for **Minimum TTL**, **Maximum TTL**, and **Default TTL** for a cache behavior. For information about the individual settings, see Minimum TTL, Maximum TTL, and Default TTL.
- To change the cache duration for an individual file, you can configure your origin to add a `Cache-Control` header with the `max-age` or `s-maxage` directive, or an `Expires` header to the file. For more information, see Use headers to control cache duration for individual objects.

For more information about how **Minimum TTL**, **Default TTL**, and **Maximum TTL** interact with the `max-age` and `s-maxage` directives and the `Expires` header field, see the section called "Specify the amount of time that CloudFront caches objects".

You can also control how long errors (for example, `404 Not Found`) stay in a CloudFront cache before CloudFront tries again to get the requested object by forwarding another request to your origin. For more information, see the section called "How CloudFront processes HTTP 4xx and 5xx status codes from your origin".

Topics

- Use headers to control cache duration for individual objects
- Serve stale (expired) content
- Specify the amount of time that CloudFront caches objects
- Add headers to your objects by using the Amazon S3 console

Use headers to control cache duration for individual objects

You can use the `Cache-Control` and `Expires` headers to control how long objects stay in the cache. Settings for **Minimum TTL**, **Default TTL**, and **Maximum TTL** also affect cache duration, but here's an overview of how headers can affect cache duration:

- With the `Cache-Control max-age` directive, you can specify how long (in seconds) that you want an object to remain in the cache before CloudFront gets the object again from the origin

server. The minimum expiration time CloudFront supports is 0 seconds. The maximum value is 100 years. Specify the value in the following format:

```
Cache-Control: max-age=seconds
```

For example, the following directive tells CloudFront to keep the associated object in the cache for 3600 seconds (one hour):

```
Cache-Control: max-age=3600
```

If you want objects to stay in CloudFront edge caches for a different duration than they stay in browser caches, you can use the `Cache-Control max-age` and `Cache-Control s-maxage` directives together. For more information, see Specify the amount of time that CloudFront caches objects.

- With the `Expires` header field, you can specify an expiration date and time using the format specified in RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1 Section 3.3.1, Full Date, for example:

  ```
  Sat, 27 Jun 2015 23:59:59 GMT
  ```

We recommend that you use the `Cache-Control max-age` directive instead of the `Expires` header field to control object caching. If you specify values both for `Cache-Control max-age` and for `Expires`, CloudFront uses only the value of `Cache-Control max-age`.

For more information, see Specify the amount of time that CloudFront caches objects.

You cannot use the HTTP `Cache-Control` or `Pragma` header fields in a `GET` request from a viewer to force CloudFront to go back to the origin server for the object. CloudFront ignores those header fields in viewer requests.

For more information about the `Cache-Control` and `Expires` header fields, see the following sections in *RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1*:

- Section 14.9 Cache Control
- Section 14.21 Expires

Serve stale (expired) content

CloudFront supports the `Stale-While-Revalidate` and `Stale-If-Error` cache control directives. You can use these directives to specify how long stale content is available for viewers.

Topics

- Stale-While-Revalidate
- Stale-If-Error
- Use both directives

`Stale-While-Revalidate`

This directive allows CloudFront to serve stale content from the cache while CloudFront asynchronously fetches a fresh version from the origin. This improves latency as viewers receive responses immediately from edge locations without having to wait for the background fetch. Fresh content is loaded in the background for future requests.

Example: Stale-While-Revalidate

CloudFront does the following when you set the `Cache-Control` header to use these directives.

```
Cache-Control: max-age=3600, stale-while-revalidate=600
```

1. CloudFront will cache a response for one hour (`max-age=3600`).
2. If a request is made after this duration, CloudFront serves the stale content, while concurrently sending a request to the origin to revalidate and refresh the cached content.
3. While the content is being revalidated, CloudFront serves the stale content up to 10 minutes (`stale-while-revalidate=600`).

> **Note**
>
> CloudFront will serve the stale content up to the value of the `stale-while-revalidate` directive or the value of the CloudFront maximum TTL, whichever is less. After the maximum TTL duration, the stale object won't be available from the edge cache, regardless of the `stale-while-revalidate` value.

`Stale-If-Error`

This directive allows CloudFront to serve stale content from the cache if the origin is unreachable or returns an error code that is between 500 and 600. This makes sure that viewers can access content even during an origin outage.

Example: Stale-If-Error

CloudFront does the following when you set the `Cache-Control` header to use these directives.

```
Cache-Control: max-age=3600, stale-if-error=86400
```

1. CloudFront caches the response for one hour (`max-age=3600`).
2. If the origin is down or returns an error after this duration, CloudFront continues to serve the stale content for up to 24 hours (`stale-if-error=86400`)
3. If you configured custom error responses, CloudFront will attempt to serve the stale content if an error is encountered within the specified `stale-if-error` duration. If the stale content is unavailable, CloudFront will then serve the custom error responses that you configured for the corresponding error status code. For more information, see Generate custom error responses.

Notes

- CloudFront will serve the stale content up to the value of the `stale-if-error` directive or the value of the CloudFront maximum TTL, whichever is less. After the maximum TTL duration, the stale object won't be available from the edge cache, regardless of the `stale-if-error` value.
- If you don't configure `stale-if-error` or custom error responses, CloudFront will return the stale object or forward the error response back to viewer, depending on whether the requested object is in the edge cache or not. For more information, see How CloudFront processes errors if you haven't configured custom error pages.

Use both directives

Both `stale-while-revalidate` and `stale-if-error` are independent cache control directives that you can use together to reduce latency and to add a buffer for your origin to respond or recover.

Example: Using both directives

CloudFront does the following when you set the `Cache-Control` header to use the following directives.

```
Cache-Control: max-age=3600, stale-while-revalidate=600, stale-if-error=86400
```

1. CloudFront caches the response for one hour (`max-age=3600`).
2. If a request is made after this duration, CloudFront serves the stale content for up to 10 minutes (`stale-while-revalidate=600`) while the content is being revalidated.
3. If the origin server returns an error while CloudFront attempts to revalidate the content, CloudFront will continue to serve the stale content for up to 24 hours (`stale-if-error=86400`).

Caching is a balance between performance and freshness. Using directives like `stale-while-revalidate` and `stale-if-error` can enhance performance and user experience, but make sure the configurations align with how fresh you want your content to be. Stale content directives are best suited for use cases where content needs to be refreshed but having the latest version is non-essential. Additionally, if your content doesn't change or rarely changes, `stale-while-revalidate` could add unnecessary network requests. Instead, consider setting a long cache duration.

Specify the amount of time that CloudFront caches objects

To control the amount of time that CloudFront keeps an object in the cache before sending another request to the origin, you can:

- Set the minimum, maximum, and default TTL values in a CloudFront distribution's cache behavior. You can set these values in a cache policy attached to the cache behavior (recommended), or in the legacy cache settings.
- Include the `Cache-Control` or `Expires` header in responses from the origin. These headers also help determine how long a browser keeps an object in the browser cache before sending another request to CloudFront.

The following table explains how the `Cache-Control` and `Expires` headers sent from the origin work together with the TTL settings in a cache behavior to affect caching.

Origin headers	Minimum TTL = 0	Minimum TTL > 0
The origin adds a `Cache-Control: max-age` directive to the object	**CloudFront caching** CloudFront caches the object for the lesser of the value of the `Cache-Control: max-age` directive or the value of the CloudFront maximum TTL. **Browser caching** Browsers cache the object for the value of the `Cache-Control: max-age` directive.	**CloudFront caching** CloudFront caching depends on the values of the CloudFront minimum TTL and maximum TTL and the `Cache-Control max-age` directive: • If minimum TTL < `max-age` < maximum TTL, then CloudFront caches the object for the value of the `Cache-Control: max-age` directive. • If `max-age` < minimum TTL, then CloudFront caches the object for the value of the CloudFront minimum TTL. • If `max-age` > maximum TTL, then CloudFront caches the object for the value of the CloudFront maximum TTL. **Browser caching** Browsers cache the object for the value of the `Cache-Control: max-age` directive.

Origin headers	Minimum TTL = 0	Minimum TTL > 0
The origin does not add a `Cache-Control: max-age` directive to the object	**CloudFront caching** CloudFront caches the object for the value of the CloudFront default TTL. **Browser caching** Depends on the browser.	**CloudFront caching** CloudFront caches the object for the greater of the value of the CloudFront minimum TTL or default TTL. **Browser caching** Depends on the browser.

Origin headers	Minimum TTL = 0	Minimum TTL > 0
The origin adds `Cache-Control: max-age` and `Cache-Control: s-maxage` directives to the object	**CloudFront caching** CloudFront caches the object for the lesser of the value of the `Cache-Control: s-maxage` directive or the value of the CloudFront maximum TTL. **Browser caching** Browsers cache the object for the value of the `Cache-Control max-age` directive.	**CloudFront caching** CloudFront caching depends on the values of the CloudFront minimum TTL and maximum TTL and the `Cache-Control: s-maxage` directive: • If minimum TTL < `s-maxage` < maximum TTL, then CloudFront caches the object for the value of the `Cache-Control: s-maxage` directive. • If `s-maxage` < minimum TTL, then CloudFront caches the object for the value of the CloudFront minimum TTL. • If `s-maxage` > maximum TTL, then CloudFront caches the object for the value of the CloudFront maximum TTL. **Browser caching** Browsers cache the object for the value of the `Cache-Control: max-age` directive.

Origin headers	Minimum TTL = 0	Minimum TTL > 0
The origin adds an `Expires` header to the object	**CloudFront caching** CloudFront caches the object until the date in the `Expires` header or for the value of the CloudFront maximum TTL, whichever is sooner. **Browser caching** Browsers cache the object until the date in the `Expires` header.	**CloudFront caching** CloudFront caching depends on the values of the CloudFront minimum TTL and maximum TTL and the `Expires` header: • If minimum TTL < `Expires` < maximum TTL, then CloudFront caches the object until the date and time in the `Expires` header. • If `Expires` < minimum TTL, then CloudFront caches the object for the value of the CloudFront minimum TTL. • If `Expires` > maximum TTL, then CloudFront caches the object for the value of the CloudFront maximum TTL. **Browser caching** Browsers cache the object until the date and time in the `Expires` header.

Origin headers	Minimum TTL = 0	Minimum TTL > 0
Origin adds `Cache-Control: no-cache`, `no-store`, and/or `private` directives to the object	CloudFront and browsers respect the headers.	**CloudFront caching** CloudFront caches the object for the value of the CloudFront minimum TTL. See the warning below this table. **Browser caching** Browsers respect the headers.

Warning

- If your minimum TTL is greater than 0, CloudFront uses the cache policy's minimum TTL, even if the `Cache-Control: no-cache`, `no-store`, and/or `private` directives are present in the origin headers.
  - If the origin is reachable, CloudFront gets the object from the origin and returns it to the viewer.
  - If the origin is unreachable and the minimum *or* maximum TTL value is greater than 0, CloudFront will serve the object that it got from the origin previously.

  To avoid this behavior, include the `Cache-Control: stale-if-error=0` directive with the object returned from the origin. This causes CloudFront to return an error in response to future requests if the origin is unreachable, rather than returning the object that it got from the origin previously.
- CloudFront doesn't cache the HTTP 501 status code (Not Implemented) from an S3 origin when the origin headers include the `Cache-Control: no-cache`, `no-store`, and/or `private` directives. This is the default behavior for an S3 origin, even if your minimum TTL setting is greater than 0.

For information about how to change settings for distributions using the CloudFront console, see Update a distribution. For information about how to change settings for distributions using the CloudFront API, see UpdateDistribution.

Add headers to your objects by using the Amazon S3 console

You can add the `Cache-Control` or `Expires` header field to your Amazon S3 objects. To do so, you modify the metadata fields for the object.

To add a `Cache-Control` or `Expires` header field to Amazon S3 objects

1. Follow the procedure in the **Replacing system-defined metadata** section on the Editing object metadata in the Amazon S3 console topic in the *Amazon S3 User Guide*.
2. For **Key**, choose the name of the header that you are adding (**Cache-Control** or **Expires**).
3. For **Value**, enter a header value. For example, for a `Cache-Control` header, you could enter `max-age=86400`. For `Expires`, you could enter an expiration date and time such as `Wed, 30 Jun 2021 09:28:00 GMT`.
4. Follow the rest of the procedure to save your metadata changes.

Cache content based on query string parameters

Some web applications use query strings to send information to the origin. A query string is the part of a web request that appears after a ? character; the string can contain one or more parameters, separated by & characters. In the following example, the query string includes two parameters, *`color=red`* and *`size=large`*:

`https://d111111abcdef8.cloudfront.net/images/image.jpg?`*`color=red&size=large`*

For distributions, you can choose if you want CloudFront to forward query strings to your origin and whether to cache your content based on all parameters or on selected parameters. Why might this be useful? Consider the following example.

Suppose that your website is available in five languages. The directory structure and file names for all five versions of the website are identical. As a user views your website, requests that are forwarded to CloudFront include a language query string parameter based on the language that the user chose. You can configure CloudFront to forward query strings to the origin and to cache based on the language parameter. If you configure your web server to return the version of a

given page that corresponds with the selected language, CloudFront caches each language version separately, based on the value of the language query string parameter.

In this example, if the main page for your website is `main.html`, the following five requests cause CloudFront to cache `main.html` five times, once for each value of the language query string parameter:

- `https://d111111abcdef8.cloudfront.net/main.html?language=de`
- `https://d111111abcdef8.cloudfront.net/main.html?language=en`
- `https://d111111abcdef8.cloudfront.net/main.html?language=es`
- `https://d111111abcdef8.cloudfront.net/main.html?language=fr`
- `https://d111111abcdef8.cloudfront.net/main.html?language=jp`

Note the following:

- Some HTTP servers don't process query string parameters and, therefore, don't return different versions of an object based on parameter values. For these origins, if you configure CloudFront to forward query string parameters to the origin, CloudFront still caches based on the parameter values even though the origin returns identical versions of the object to CloudFront for every parameter value.
- For query string parameters to work as described in the example above with the languages, you must use the & character as the delimiter between query string parameters. If you use a different delimiter, you might get unexpected results, depending on which parameters you specify for CloudFront to use as a basis for caching, and the order in which the parameters appear in the query string.

  The following examples show what happens if you use a different delimiter and you configure CloudFront to cache based only on the `color` parameter:

  - In the following request, CloudFront caches your content based on the value of the `color` parameter, but CloudFront interprets the value as *`red;size=large`*:

    `https://d111111abcdef8.cloudfront.net/images/image.jpg?color=red;size=large`

  - In the following request, CloudFront caches your content but doesn't base caching on the query string parameters. This is because you configured CloudFront to cache based on the

`color` parameter, but CloudFront interprets the following string as containing only a `size` parameter that has a value of *`large;color=red`*:

```
https://d111111abcdef8.cloudfront.net/images/
image.jpg?size=large;color=red
```

You can configure CloudFront to do one of the following:

- Don't forward query strings to the origin at all. If you don't forward query strings, CloudFront doesn't cache based on query string parameters.
- Forward query strings to the origin, and cache based on all parameters in the query string.
- Forward query strings to the origin, and cache based on specified parameters in the query string.

For more information, see the section called “Optimize caching”.

Topics

- Console and API settings for query string forwarding and caching
- Optimize caching
- Query string parameters and CloudFront standard logs (access logs)

Console and API settings for query string forwarding and caching

When you create a distribution in the CloudFront console, CloudFront configures query string forwarding and caching for you based on your origin type. Optionally, you can manually edit these settings. For more information, see the following settings in the the section called “All distribution settings”:

- the section called “Query string forwarding and caching”
- the section called “Query string allowlist”

To configure query string forwarding and caching with the CloudFront API, see CachePolicy and OriginRequestPolicy in the *Amazon CloudFront API Reference*.

Optimize caching

When you configure CloudFront to cache based on query string parameters, you can take the following steps to reduce the number of requests that CloudFront forwards to your origin. When CloudFront edge locations serve objects, you reduce the load on your origin server and reduce latency because objects are served from locations that are closer to your users.

Cache based only on parameters for which your origin returns different versions of an object

For each query string parameter that your web application forwards to CloudFront, CloudFront forwards requests to your origin for every parameter value and caches a separate version of the object for every parameter value. This is true even if your origin always returns the same object regardless of the parameter value. For multiple parameters, the number of requests and the number of objects multiply.

We recommend that you configure CloudFront to cache based only on the query string parameters for which your origin returns different versions, and that you carefully consider the merits of caching based on each parameter. For example, suppose you have a retail website. You have pictures of a jacket in six different colors, and the jacket comes in 10 different sizes. The pictures that you have of the jacket show the different colors but not the different sizes. To optimize caching, you should configure CloudFront to cache based only on the color parameter, not on the size parameter. This increases the likelihood that CloudFront can serve a request from the cache, which improves performance and reduces the load on your origin.

Always list parameters in the same order

The order of parameters matters in query strings. In the following example, the query strings are identical except that the parameters are in a different order. This causes CloudFront to forward two separate requests for image.jpg to your origin and to cache two separate versions of the object:

- `https://d111111abcdef8.cloudfront.net/images/image.jpg?color=red&size=large`
- `https://d111111abcdef8.cloudfront.net/images/image.jpg?size=large&color=red`

We recommend that you always list parameter names in the same order, such as alphabetical order.

Always use the same case for parameter names and values

CloudFront considers the case of parameter names and values when caching based on query string parameters. In the following example, the query strings are identical except for the case of parameter names and values. This causes CloudFront to forward four separate requests for image.jpg to your origin and to cache four separate versions of the object:

- `https://d111111abcdef8.cloudfront.net/images/image.jpg?color=red`
- `https://d111111abcdef8.cloudfront.net/images/image.jpg?color=Red`
- `https://d111111abcdef8.cloudfront.net/images/image.jpg?Color=red`
- `https://d111111abcdef8.cloudfront.net/images/image.jpg?Color=Red`

We recommend that you use case consistently for parameter names and values, such as all lowercase.

Don't use parameter names that conflict with signed URLs

If you're using signed URLs to restrict access to your content (if you added trusted signers to your distribution), CloudFront removes the following query string parameters before forwarding the rest of the URL to your origin:

- `Expires`
- `Key-Pair-Id`
- `Policy`
- `Signature`

If you're using signed URLs and you want to configure CloudFront to forward query strings to your origin, your own query string parameters cannot be named `Expires`, `Key-Pair-Id`, `Policy`, or `Signature`.

Query string parameters and CloudFront standard logs (access logs)

If you enable logging, CloudFront logs the full URL, including query string parameters. This is true regardless of whether you have configured CloudFront to forward query strings to the origin. For more information about CloudFront logging, see the section called "Access logs (standard logs)".

Cache content based on cookies

By default, CloudFront doesn't consider cookies when processing requests and responses, or when caching your objects in edge locations. If CloudFront receives two requests that are identical except for what's in the `Cookie` header, then, by default, CloudFront treats the requests as identical and returns the same object for both requests.

You can configure CloudFront to forward to your origin some or all of the cookies in viewer requests, and to cache separate versions of your objects based on the cookie values that it forwards. When you do this, CloudFront uses some or all of the cookies in viewer requests—whichever ones it's configured to forward—to uniquely identify an object in the cache.

For example, suppose that requests for `locations.html` contain a `country` cookie that has a value of either `uk` or `fr`. When you configure CloudFront to cache your objects based on the value of the `country` cookie, CloudFront forwards requests for `locations.html` to the origin and includes the `country` cookie and its value. Your origin returns `locations.html`, and CloudFront caches the object once for requests in which the value of the `country` cookie is `uk` and once for requests in which the value is `fr`.

Important

Amazon S3 and some HTTP servers don't process cookies. Don't configure CloudFront to forward cookies to an origin that doesn't process cookies or doesn't vary its response based on cookies. That can cause CloudFront to forward more requests to the origin for the same object, which slows performance and increases the load on the origin. If, considering the previous example, your origin doesn't process the `country` cookie or always returns the same version of `locations.html` to CloudFront regardless of the value of the `country` cookie, don't configure CloudFront to forward that cookie.

Conversely, if your custom origin depends on a particular cookie or sends different responses based on a cookie, make sure you configure CloudFront to forward that cookie to the origin. Otherwise, CloudFront removes the cookie before forwarding the request to your origin.

To configure cookie forwarding, you update your distribution's cache behavior. For more information about cache behaviors, see Cache behavior settings, particularly the Forward cookies and Allowlist cookies sections.

You can configure each cache behavior to do one of the following:

- **Forward all cookies to your origin –** CloudFront includes all cookies sent by the viewer when it forwards requests to the origin. When your origin returns a response, CloudFront caches the response using the cookie names and values in the viewer request. If the origin response includes `Set-Cookie` headers, CloudFront returns them to the viewer with the requested object. CloudFront also caches the `Set-Cookie` headers with the object returned from the origin, and sends those `Set-Cookie` headers to viewers on all cache hits.
- **Forward a set of cookies that you specify –** CloudFront removes any cookies that the viewer sends that aren't on the allowlist before it forwards a request to the origin. CloudFront caches the response using the listed cookies names and values in the viewer request. If the origin response includes `Set-Cookie` headers, CloudFront returns them to the viewer with the requested object. CloudFront also caches the `Set-Cookie` headers with the object returned from the origin, and sends those `Set-Cookie` headers to viewers on all cache hits.

  For information about specifying wildcards in cookie names, see Allowlist cookies.

  For the current quota on the number of cookie names that you can forward for each cache behavior, or to request a higher quota, see Quotas on query strings (legacy cache settings).
- **Don't forward cookies to your origin –** CloudFront doesn't cache your objects based on cookie sent by the viewer. In addition, CloudFront removes cookies before forwarding requests to your origin, and removes `Set-Cookie` headers from responses before returning responses to your viewers. Because this isn't an optimal way to use your origin resources, when you select this cache behavior, you should make sure that your origin doesn't include cookies in origin responses by default.

Note the following about specifying the cookies that you want to forward:

Access logs

If you configure CloudFront to log requests and to log cookies, CloudFront logs all cookies and all cookie attributes, even if you configure CloudFront not to forward cookies to your origin or if you configure CloudFront to forward only specific cookies. For more information about CloudFront logging, see Access logs (standard logs).

Case sensitivity

Cookie names and values are both case-sensitive. For example, if CloudFront is configured to forward all cookies, and two viewer requests for the same object have cookies that are identical except for case, CloudFront caches the object twice.

CloudFront sorts cookies

If CloudFront is configured to forward cookies (all or a subset), CloudFront sorts the cookies in natural order by cookie name before forwarding the request to your origin.

> **Note**
>
> Cookie names that start with the $ character are unsupported. CloudFront will remove the cookie before forwarding the request to the origin. You can remove the $ character or specify a different character at the start of the cookie name.

`If-Modified-Since` and `If-None-Match`

`If-Modified-Since` and `If-None-Match` conditional requests are not supported when CloudFront is configured to forward cookies (all or a subset).

Standard name–value pair format is required

CloudFront forwards a cookie header only if the value conforms to the standard name–value pair format, for example: `"Cookie: cookie1=value1; cookie2=value2"`

Disable caching of `Set-Cookie` headers

If CloudFront is configured to forward cookies to the origin (whether all or specific cookies), it also caches the `Set-Cookie` headers received in the origin response. CloudFront includes these `Set-Cookie` headers in its response to the original viewer, and also includes them in subsequent responses that are served from the CloudFront cache.

If you want to receive cookies at your origin but you don't want CloudFront to cache the `Set-Cookie` headers in your origin's responses, configure your origin to add a `Cache-Control` header with a `no-cache` directive that specifies `Set-Cookie` as a field name. For example: `Cache-Control: no-cache="Set-Cookie"`. For more information, see Response Cache-Control Directives in the *Hypertext Transfer Protocol (HTTP/1.1): Caching* standard.

Maximum length of cookie names

If you configure CloudFront to forward specific cookies to your origin, the total number of bytes in all of the cookie names that you configure CloudFront to forward can't exceed 512 minus the number of cookies that you're forwarding. For example, if you configure CloudFront to forward 10 cookies to your origin, the combined length of the names of the 10 cookies can't exceed 502 bytes (512 – 10).

If you configure CloudFront to forward all cookies to your origin, the length of cookie names doesn't matter.

For information about using the CloudFront console to update a distribution so CloudFront forwards cookies to the origin, see Update a distribution. For information about using the CloudFront API to update a distribution, see UpdateDistribution in the *Amazon CloudFront API Reference*.

Cache content based on request headers

With CloudFront, you can choose whether you want CloudFront to forward headers to your origin and to cache separate versions of a specified object based on the header values in viewer requests. With this, you can serve different versions of your content based on the device the user is using, the location of the viewer, the language the viewer is using, and a variety of other criteria.

Topics

- Headers and distributions – overview
- Select the headers to base caching on
- Configure CloudFront to respect CORS settings
- Configure caching based on the device type
- Configure caching based on the language of the viewer
- Configure caching based on the location of the viewer
- Configure caching based on the protocol of the request
- Configure caching for compressed files
- How caching based on headers affects performance
- How the case of headers and header values affects caching
- Headers that CloudFront returns to the viewer

Headers and distributions – overview

By default, CloudFront doesn't consider headers when caching your objects in edge locations. If your origin returns two objects and they differ only by the values in the request headers, CloudFront caches only one version of the object.

You can configure CloudFront to forward headers to the origin, which causes CloudFront to cache multiple versions of an object based on the values in one or more request headers. To configure CloudFront to cache objects based on the values of specific headers, you specify cache behavior settings for your distribution. For more information, see Cache Based on Selected Request Headers.

For example, suppose viewer requests for `logo.jpg` contain a custom `Product` header that has a value of either `Acme` or `Apex`. When you configure CloudFront to cache your objects based on the value of the `Product` header, CloudFront forwards requests for `logo.jpg` to the origin and includes the `Product` header and header values. CloudFront caches `logo.jpg` once for requests in which the value of the `Product` header is `Acme` and once for requests in which the value is `Apex`.

You can configure each cache behavior in a distribution to do one of the following:

- Forward all headers to your origin

  > **Note**
  >
  > **For legacy cache settings** – If you configure CloudFront to forward all headers to your origin, CloudFront doesn't cache the objects associated with this cache behavior. Instead, it sends every request to the origin.

- Forward a list of headers that you specify. CloudFront caches your objects based on the values in all of the specified headers. CloudFront also forwards the headers that it forwards by default, but it caches your objects based only on the headers that you specify.
- Forward only the default headers. In this configuration, CloudFront doesn't cache your objects based on the values in the request headers.

For the current quota on the number of headers that you can forward for each cache behavior or to request a higher quota, see Quotas on headers.

For information about using the CloudFront console to update a distribution so CloudFront forwards headers to the origin, see Update a distribution. For information about using the

CloudFront API to update an existing distribution, see Update Distribution in the *Amazon CloudFront API Reference*.

Select the headers to base caching on

The headers that you can forward to the origin and that CloudFront bases caching on depend on whether your origin is an Amazon S3 bucket or a custom origin.

- **Amazon S3 –** You can configure CloudFront to forward and to cache your objects based on a number of specific headers (see the following list of exceptions). However, we recommend that you avoid forwarding headers with an Amazon S3 origin unless you need to implement cross-origin resource sharing (CORS) or you want to personalize content by using Lambda@Edge in origin-facing events.
  - To configure CORS, you must forward headers that allow CloudFront to distribute content for websites that are enabled for cross-origin resource sharing (CORS). For more information, see Configure CloudFront to respect CORS settings.
  - To personalize content by using headers that you forward to your Amazon S3 origin, you write and add Lambda@Edge functions and associate them with your CloudFront distribution to be triggered by an origin-facing event. For more information about working with headers to personalize content, see Personalize content by country or device type headers - examples.

    We recommend that you avoid forwarding headers that you aren't using to personalize content because forwarding extra headers can reduce your cache hit ratio. That is, CloudFront can't serve as many requests from edge caches, as a proportion of all requests.
- **Custom origin** – You can configure CloudFront to cache based on the value of any request header except the following:
  - `Connection`
  - `Cookie` – If you want to forward and cache based on cookies, you use a separate setting in your distribution. For more information, see Cache content based on cookies.
  - `Host (for Amazon S3 origins)`
  - `Proxy-Authorization`
  - `TE`
  - `Upgrade`

  You can configure CloudFront to cache objects based on values in the `Date` and `User-Agent` headers, but we don't recommend it. These headers have numerous possible values, and caching

based on their values could cause CloudFront to forward significantly more requests to your origin.

For a full list of HTTP request headers and how CloudFront processes them, see HTTP request headers and CloudFront behavior (custom and Amazon S3 origins).

Configure CloudFront to respect CORS settings

If you have enabled cross-origin resource sharing (CORS) on an Amazon S3 bucket or a custom origin, you must choose specific headers to forward, to respect the CORS settings. The headers that you must forward differ depending on the origin (Amazon S3 or custom) and whether you want to cache `OPTIONS` responses.

Amazon S3

- If you want `OPTIONS` responses to be cached, do the following:
  - Choose the options for default cache behavior settings that enable caching for `OPTIONS` responses.
  - Configure CloudFront to forward the following headers: `Origin`, `Access-Control-Request-Headers`, and `Access-Control-Request-Method`.
- If you don't want `OPTIONS` responses to be cached, configure CloudFront to forward the `Origin` header, together with any other headers required by your origin (for example, `Access-Control-Request-Headers`, `Access-Control-Request-Method`, or others).

Custom origins – Forward the `Origin` header along with any other headers required by your origin.

To configure CloudFront to cache responses based on CORS, you must configure CloudFront to forward headers by using a cache policy. For more information, see *Control the cache key with a policy*.

For more information about CORS and Amazon S3, see Using cross-origin resource sharing (CORS) in the *Amazon Simple Storage Service User Guide*.

Configure caching based on the device type

If you want CloudFront to cache different versions of your objects based on the device a user is using to view your content, configure CloudFront to forward the applicable headers to your custom origin:

- `CloudFront-Is-Desktop-Viewer`
- `CloudFront-Is-Mobile-Viewer`
- `CloudFront-Is-SmartTV-Viewer`
- `CloudFront-Is-Tablet-Viewer`

Based on the value of the `User-Agent` header, CloudFront sets the value of these headers to `true` or `false` before forwarding the request to your origin. If a device falls into more than one category, more than one value might be `true`. For example, for some tablet devices, CloudFront might set both `CloudFront-Is-Mobile-Viewer` and `CloudFront-Is-Tablet-Viewer` to `true`.

Configure caching based on the language of the viewer

If you want CloudFront to cache different versions of your objects based on the language specified in the request, configure CloudFront to forward the `Accept-Language` header to your origin.

Configure caching based on the location of the viewer

If you want CloudFront to cache different versions of your objects based on the country that the request came from, configure CloudFront to forward the `CloudFront-Viewer-Country` header to your origin. CloudFront automatically converts the IP address that the request came from into a two-letter country code. For an easy-to-use list of country codes, sortable by code and by country name, see the Wikipedia entry ISO 3166-1 alpha-2.

Configure caching based on the protocol of the request

If you want CloudFront to cache different versions of your objects based on the protocol of the request, HTTP or HTTPS, configure CloudFront to forward the `CloudFront-Forwarded-Proto` header to your origin.

Configure caching for compressed files

If your origin supports Brotli compression, you can cache based on the `Accept-Encoding` header. Configure caching based on `Accept-Encoding` only if your origin serves different content based on the header.

How caching based on headers affects performance

When you configure CloudFront to cache based on one or more headers and the headers have more than one possible value, CloudFront forwards more requests to your origin server for the same object. This slows performance and increases the load on your origin server. If your origin server returns the same object regardless of the value of a given header, we recommend that you don't configure CloudFront to cache based on that header.

If you configure CloudFront to forward more than one header, the order of the headers in viewer requests doesn't affect caching as long as the values are the same. For example, if one request contains the headers A:1,B:2 and another request contains B:2,A:1, CloudFront caches just one copy of the object.

How the case of headers and header values affects caching

When CloudFront caches based on header values, it doesn't consider the case of the header name, but it does consider the case of the header value:

- If viewer requests include both `Product:Acme` and `product:Acme`, CloudFront caches an object only once. The only difference between them is the case of the header name, which doesn't affect caching.
- If viewer requests include both `Product:Acme` and `Product:acme`, CloudFront caches an object twice, because the value is `Acme` in some requests and `acme` in others.

Headers that CloudFront returns to the viewer

Configuring CloudFront to forward and cache headers does not affect which headers CloudFront returns to the viewer. CloudFront returns all of the headers that it gets from the origin with a few exceptions. For more information, see the applicable topic:

- **Amazon S3 origins –** See HTTP response headers that CloudFront removes or updates.
- **Custom origins –** See HTTP response headers that CloudFront removes or replaces.

Control the cache key with a policy

With a CloudFront *cache policy*, you can specify the HTTP headers, cookies, and query strings that CloudFront includes in the *cache key* for objects that are cached at CloudFront edge locations. The cache key is the unique identifier for every object in the cache, and it determines whether a viewer's HTTP request results in a *cache hit*.

A cache hit occurs when a viewer request generates the same cache key as a prior request, and the object for that cache key is in the edge location's cache and valid. When there's a cache hit, the object is served to the viewer from a CloudFront edge location, which has the following benefits:

- Reduced load on your origin server
- Reduced latency for the viewer

Including fewer values in the cache key increases the likelihood of a cache hit. This can get you better performance from your website or application because there's a higher *cache hit ratio* (a higher proportion of viewer requests that result in a cache hit). For more information, see Understand the cache key.

To control the cache key, you use a CloudFront *cache policy*. You attach a cache policy to one or more cache behaviors in a CloudFront distribution.

You can also use the cache policy to specify time to live (TTL) settings for objects in the CloudFront cache, and enable CloudFront to request and cache compressed objects.

> **Note**
>
> Cache settings don't affect gRPC requests because gRPC traffic can't be cached. For more information, see Using gRPC with CloudFront distributions.

Topics

- Understand cache policies
- Create cache policies
- Use managed cache policies
- Understand the cache key

Understand cache policies

You can use a cache policy to improve your cache hit ratio by controlling the values (URL query strings, HTTP headers, and cookies) that are included in the cache key. CloudFront provides some predefined cache policies, known as *managed policies*, for common use cases. You can use these managed policies, or you can create your own cache policy that's specific to your needs. For more information about the managed policies, see Use managed cache policies.

A cache policy contains the following settings, which are categorized into *policy information*, *time to live (TTL) settings*, and *cache key settings*.

Policy information

Name

A name to identify the cache policy. In the console, you use the name to attach the cache policy to a cache behavior.

Description

A comment to describe the cache policy. This is optional, but it can help you identify the purpose of the cache policy.

Time to live (TTL) settings

The time to live (TTL) settings work together with the `Cache-Control` and `Expires` HTTP headers (if they're in the origin response) to determine how long objects in the CloudFront cache remain valid.

Minimum TTL

The minimum amount of time, in seconds, that you want objects to stay in the CloudFront cache before CloudFront checks with the origin to see if the object has been updated. For more information, see Manage how long content stays in the cache (expiration).

> **Warning**
>
> If your minimum TTL is greater than 0, CloudFront will cache content for at least the duration specified in the cache policy's minimum TTL, even if the `Cache-Control: no-cache`, `no-store`, or `private` directives are present in the origin headers.

Maximum TTL

The maximum amount of time, in seconds, that objects stay in the CloudFront cache before CloudFront checks with the origin to see if the object has been updated. CloudFront uses this setting only when the origin sends `Cache-Control` or `Expires` headers with the object. For more information, see Manage how long content stays in the cache (expiration).

Default TTL

The default amount of time, in seconds, that you want objects to stay in the CloudFront cache before CloudFront checks with the origin to see if the object has been updated. CloudFront uses this setting's value as the object's TTL only when the origin does *not* send `Cache-Control` or `Expires` headers with the object. For more information, see Manage how long content stays in the cache (expiration).

> **Note**
>
> If the **Minimum TTL**, **Maximum TTL**, and **Default TTL** settings are all set to 0, this disables CloudFront caching.

Cache key settings

Cache key settings specify the values in viewer requests that CloudFront includes in the cache key. The values can include URL query strings, HTTP headers, and cookies. The values that you include in the cache key are automatically included in requests that CloudFront sends to the origin, known as *origin requests*. For information about controlling origin requests without affecting the cache key, see Control origin requests with a policy.

Cache key settings include:

- Headers

- Cookies
- Query strings
- Compression support

Headers

The HTTP headers in viewer requests that CloudFront includes in the cache key and in origin requests. For headers, you can choose one of the following settings:

- **None** – The HTTP headers in viewer requests are *not* included in the cache key and are *not* automatically included in origin requests.
- **Include the following headers** – You specify which of the HTTP headers in viewer requests are included in the cache key and automatically included in origin requests.

When you use the **Include the following headers** setting, you specify HTTP headers by their name, not their value. For example, consider the following HTTP header:

```
Accept-Language: en-US,en;q=0.5
```

In this case, you specify the header as `Accept-Language`, not as `Accept-Language: en-US,en;q=0.5`. However, CloudFront includes the full header, including its value, in the cache key and in origin requests.

You can also include certain headers generated by CloudFront in the cache key. For more information, see the section called "Add CloudFront request headers".

Cookies

The cookies in viewer requests that CloudFront includes in the cache key and in origin requests. For cookies, you can choose one of the following settings:

- **None** – The cookies in viewer requests are *not* included in the cache key and are *not* automatically included in origin requests.
- **All** – All cookies in viewer requests are included in the cache key and are automatically included in origin requests.
- **Include specified cookies** – You specify which of the cookies in viewer requests are included in the cache key and automatically included in origin requests.
- **Include all cookies except** – You specify which of the cookies in viewer requests are *not* included in the cache key and are *not* automatically included in origin requests. All other

cookies, except for the ones you specify, *are* included in the cache key and automatically included in origin requests.

When you use the **Include specified cookies** or **Include all cookies except** setting, you specify cookies by their name, not their value. For example, consider the following `Cookie` header:

```
Cookie: session_ID=abcd1234
```

In this case, you specify the cookie as `session_ID`, not as `session_ID=abcd1234`. However, CloudFront includes the full cookie, including its value, in the cache key and in origin requests.

You can use the asterisk (*) as a wildcard in cookie names. It matches 0 or more characters in the cookie name. The wildcard applies to both the **Include specified cookies** and **Include all cookies except** settings.

For example, if you specify `userid_*`, CloudFront matches all of the cookies whose names begin with `userid_`, such as `userid_1234` and `userid_5678`. A cookie name can't consist only of asterisks. To include all cookies, use the **All** setting instead.

Query strings

The URL query strings in viewer requests that CloudFront includes in the cache key and in origin requests. For query strings, you can choose one of the following settings:

- **None** – The query strings in viewer requests are *not* included in the cache key and are *not* automatically included in origin requests.
- **All** – All query strings in viewer requests are included in the cache key and are also automatically included in origin requests.
- **Include specified query strings** – You specify which of the query strings in viewer requests are included in the cache key and automatically included in origin requests.
- **Include all query strings except** – You specify which of the query strings in viewer requests are *not* included in the cache key and are *not* automatically included in origin requests. All other query strings, except for the ones you specify, *are* included in the cache key and automatically included in origin requests.

When you use the **Include specified query strings** or **Include all query strings except** setting, you specify query strings by their name, not their value. For example, consider the following URL path:

```
/content/stories/example-story.html?split-pages=false
```

In this case, you specify the query string as `split-pages`, not as `split-pages=false`. However, CloudFront includes the full query string, including its value, in the cache key and in origin requests.

> **Note**
>
> For the header and query string cache key settings, CloudFront treats the asterisk character (*) as a literal string, not as a wildcard. For cookies, the asterisk character is a wildcard. For more information about cookies, see Cookies.

Compression support

These settings enable CloudFront to request and cache objects that are compressed in the Gzip or Brotli compression formats, when the viewer supports it. These settings also allow CloudFront compression to work. Viewers indicate their support for these compression formats with the `Accept-Encoding` HTTP header.

> **Note**
>
> The Chrome and Firefox web browsers support Brotli compression only when the request is sent using HTTPS. These browsers do not support Brotli with HTTP requests.

Enable these settings when any of the following are true:

- Your origin returns Gzip compressed objects when viewers support them (requests contain the `Accept-Encoding` HTTP header with `gzip` as a value). In this case, use the **Gzip enabled** setting (set `EnableAcceptEncodingGzip` to `true` in the CloudFront API, AWS SDKs, AWS CLI, or CloudFormation).
- Your origin returns Brotli compressed objects when viewers support them (requests contain the `Accept-Encoding` HTTP header with `br` as a value). In this case, use the **Brotli enabled** setting (set `EnableAcceptEncodingBrotli` to `true` in the CloudFront API, AWS SDKs, AWS CLI, or CloudFormation).
- The cache behavior that this cache policy is attached to is configured with CloudFront compression. In this case, you can enable caching for either Gzip or Brotli, or both. When CloudFront compression is enabled, enabling caching for both formats can help to reduce your costs for data transfer out to the internet.

> **Note**
>
> If you enable caching for one or both of these compression formats, do not include the `Accept-Encoding` header in an origin request policy that's associated with the same cache behavior. CloudFront always includes this header in origin requests when caching is enabled for either of these formats, so including `Accept-Encoding` in an origin request policy has no effect.

If your origin server does not return Gzip or Brotli compressed objects, or the cache behavior is not configured with CloudFront compression, don't enable caching for compressed objects. If you do, it might cause a decrease in your cache hit ratio.

The following explains how these settings affect a CloudFront distribution. All of the following scenarios assume that the viewer request includes the `Accept-Encoding` header. When the viewer request does not include the `Accept-Encoding` header, CloudFront doesn't include this header in the cache key and doesn't include it in the corresponding origin request.

When caching compressed objects is enabled for both compression formats

If the viewer supports both Gzip and Brotli—that is, if the `gzip` and `br` values are both in the `Accept-Encoding` header in the viewer request—CloudFront does the following:

- Normalizes the header to `Accept-Encoding: br,gzip` and includes the normalized header in the cache key. The cache key doesn't include other values that were in the `Accept-Encoding` header sent by the viewer.
- If the edge location has a Brotli or Gzip compressed object in the cache that matches the request and is not expired, the edge location returns the object to the viewer.
- If the edge location doesn't have a Brotli or Gzip compressed object in the cache that matches the request and is not expired, CloudFront includes the normalized header (`Accept-Encoding: br,gzip`) in the corresponding origin request. The origin request doesn't include other values that were in the `Accept-Encoding` header sent by the viewer.

If the viewer supports one compression format but not the other—for example, if `gzip` is a value in the `Accept-Encoding` header in the viewer request but `br` is not—CloudFront does the following:

- Normalizes the header to `Accept-Encoding: gzip` and includes the normalized header in the cache key. The cache key doesn't include other values that were in the `Accept-Encoding` header sent by the viewer.
- If the edge location has a Gzip compressed object in the cache that matches the request and is not expired, the edge location returns the object to the viewer.
- If the edge location doesn't have a Gzip compressed object in the cache that matches the request and is not expired, CloudFront includes the normalized header (`Accept-Encoding: gzip`) in the corresponding origin request. The origin request doesn't include other values that were in the `Accept-Encoding` header sent by the viewer.

To understand what CloudFront does if the viewer supports Brotli but not Gzip, replace the two compression formats with each other in the preceding example.

If the viewer does not support Brotli or Gzip—that is, the `Accept-Encoding` header in the viewer request does not contain `br` or `gzip` as values—CloudFront:

- Doesn't include the `Accept-Encoding` header in the cache key.
- Includes `Accept-Encoding: identity` in the corresponding origin request. The origin request doesn't include other values that were in the `Accept-Encoding` header sent by the viewer.

When caching compressed objects is enabled for one compression format, but not the other

If the viewer supports the format for which caching is enabled—for example, if caching compressed objects is enabled for Gzip and the viewer supports Gzip (`gzip` is one of the values in the `Accept-Encoding` header in the viewer request)—CloudFront does the following:

- Normalizes the header to `Accept-Encoding: gzip` and includes the normalized header in the cache key.
- If the edge location has a Gzip compressed object in the cache that matches the request and is not expired, the edge location returns the object to the viewer.
- If the edge location doesn't have a Gzip compressed object in the cache that matches the request and is not expired, CloudFront includes the normalized header (`Accept-Encoding: gzip`) in the corresponding origin request. The origin request doesn't include other values that were in the `Accept-Encoding` header sent by the viewer.

This behavior is the same when the viewer supports both Gzip and Brotli (the `Accept-Encoding` header in the viewer request includes both `gzip` *and* `br` as values), because in this scenario, caching compressed objects for Brotli is not enabled.

To understand what CloudFront does if caching compressed objects is enabled for Brotli but not Gzip, replace the two compression formats with each other in the preceding example.

If the viewer does not support the compression format for which caching is enabled (the `Accept-Encoding` header in the viewer request doesn't contain the value for that format), CloudFront:

- Doesn't include the `Accept-Encoding` header in the cache key.
- Includes `Accept-Encoding: identity` in the corresponding origin request. The origin request doesn't include other values that were in the `Accept-Encoding` header sent by the viewer.

When caching compressed objects is disabled for both compression formats

When caching compressed objects is disabled for both compression formats, CloudFront treats the `Accept-Encoding` header the same as any other HTTP header in the viewer request. By default, it's not included in the cache key and it's not included in origin requests. You can include it in the headers list in a cache policy or an origin request policy the same as any other HTTP header.

Create cache policies

You can use a cache policy to improve your cache hit ratio by controlling the values (URL query strings, HTTP headers, and cookies) that are included in the cache key. You can create a cache policy in the CloudFront console, with the AWS Command Line Interface (AWS CLI), or with the CloudFront API.

After you create a cache policy, you attach it to one or more cache behaviors in a CloudFront distribution.

Console

To create a cache policy (console)

1. Sign in to the AWS Management Console and open the **Policies** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home?#/policies.
2. Choose **Create cache policy**.

3. Choose the desired setting for this cache policy. For more information, see Understand cache policies.
4. When finished, choose **Create**.

After you create a cache policy, you can attach it to a cache behavior.

To attach a cache policy to an existing distribution (console)

1. Open the **Distributions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/distributions.
2. Choose the distribution to update, then choose the **Behaviors** tab.
3. Choose the cache behavior to update, then choose **Edit**.

   Or, to create a new cache behavior, choose **Create behavior**.
4. In the **Cache key and origin requests** section, make sure that **Cache policy and origin request policy** is chosen.
5. For **Cache policy**, choose the cache policy to attach to this cache behavior.
6. At the bottom of the page, choose **Save changes**.

To attach a cache policy to a new distribution (console)

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **Create distribution**.
3. In the **Cache key and origin requests** section, make sure that **Cache policy and origin request policy** is chosen.
4. For **Cache policy**, choose the cache policy to attach to this distribution's default cache behavior.
5. Choose the desired settings for the origin, default cache behavior, and other distribution settings. For more information, see All distribution settings reference.
6. When finished, choose **Create distribution**.

CLI

To create a cache policy with the AWS Command Line Interface (AWS CLI), use the **aws cloudfront create-cache-policy** command. You can use an input file to provide the command's input parameters, rather than specifying each individual parameter as command line input.

To create a cache policy (CLI with input file)

1. Use the following command to create a file named `cache-policy.yaml` that contains all of the input parameters for the **create-cache-policy** command.

   ```
   aws cloudfront create-cache-policy --generate-cli-skeleton yaml-input > cache-policy.yaml
   ```

2. Open the file named `cache-policy.yaml` that you just created. Edit the file to specify the cache policy settings that you want, then save the file. You can remove optional fields from the file, but don't remove the required fields.

   For more information about the cache policy settings, see Understand cache policies.

3. Use the following command to create the cache policy using input parameters from the `cache-policy.yaml` file.

   ```
   aws cloudfront create-cache-policy --cli-input-yaml file://cache-policy.yaml
   ```

   Make note of the `Id` value in the command's output. This is the cache policy ID, and you need it to attach the cache policy to a CloudFront distribution's cache behavior.

To attach a cache policy to an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to update. Replace *`distribution_ID`* with the distribution's ID.

   ```
   aws cloudfront get-distribution-config --id distribution_ID --output yaml >
    dist-config.yaml
   ```

2. Open the file named `dist-config.yaml` that you just created. Edit the file, making the following changes to each cache behavior that you are updating to use a cache policy.

- In the cache behavior, add a field named `CachePolicyId`. For the field's value, use the cache policy ID that you noted after creating the policy.
- Remove the `MinTTL`, `MaxTTL`, `DefaultTTL`, and `ForwardedValues` fields from the cache behavior. These settings are specified in the cache policy, so you can't include these fields and a cache policy in the same cache behavior.
- Rename the `ETag` field to `IfMatch`, but don't change the field's value.

Save the file when finished.

3. Use the following command to update the distribution to use the cache policy. Replace *`distribution_ID`* with the distribution's ID.

```
aws cloudfront update-distribution --id distribution_ID --cli-input-yaml file://
dist-config.yaml
```

To attach a cache policy to a new distribution (CLI with input file)

1. Use the following command to create a file named `distribution.yaml` that contains all of the input parameters for the **create-distribution** command.

```
aws cloudfront create-distribution --generate-cli-skeleton yaml-input >
 distribution.yaml
```

2. Open the file named `distribution.yaml` that you just created. In the default cache behavior, in the `CachePolicyId` field, enter the cache policy ID that you noted after creating the policy. Continue editing the file to specify the distribution settings that you want, then save the file when finished.

   For more information about the distribution settings, see All distribution settings reference.

3. Use the following command to create the distribution using input parameters from the `distribution.yaml` file.

```
aws cloudfront create-distribution --cli-input-yaml file://distribution.yaml
```

API

To create a cache policy with the CloudFront API, use CreateCachePolicy. For more information about the fields that you specify in this API call, see Understand cache policies and the API reference documentation for your AWS SDK or other API client.

After you create a cache policy, you can attach it to a cache behavior, using one of the following API calls:

- To attach it to a cache behavior in an existing distribution, use UpdateDistribution.
- To attach it to a cache behavior in a new distribution, use CreateDistribution.

For both of these API calls, provide the cache policy's ID in the `CachePolicyId` field, inside a cache behavior. For more information about the other fields that you specify in these API calls, see All distribution settings reference and the API reference documentation for your AWS SDK or other API client.

Use managed cache policies

CloudFront provides a set of managed cache policies that you can attach to any of your distribution's cache behaviors. With a managed cache policy, you don't need to write or maintain your own cache policy. The managed policies use settings that are optimized for specific use cases.

To use a managed cache policy, you attach it to a cache behavior in your distribution. The process is the same as when you create a cache policy, but instead of creating a new one, you just attach one of the managed cache policies. You attach the policy either by name (with the console) or by ID (with the AWS CLI or SDKs). The names and IDs are listed in the following section.

For more information, see Create cache policies.

The following topics describe the managed cache policies that you can use.

Topics

- Amplify
- CachingDisabled
- CachingOptimized
- CachingOptimizedForUncompressedObjects

- Elemental-MediaPackage
- UseOriginCacheControlHeaders
- UseOriginCacheControlHeaders-QueryStrings

Amplify

View this policy in the CloudFront console

This policy is designed for use with an origin that is an AWS Amplify web app.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`2e54312d-136d-493c-8eb9-b001f22f67d2`

This policy has the following settings:

- **Minimum TTL:** 2 seconds
- **Maximum TTL:** 600 seconds (10 minutes)
- **Default TTL:** 2 seconds
- **Headers included in cache key:**
  - `Authorization`
  - `CloudFront-Viewer-Country`
  - `Host`

  The normalized `Accept-Encoding` header is also included because the cache compressed objects setting is enabled. For more information, see Compression support.
- **Cookies included in cache key:** All cookies are included.
- **Query strings included in cache key:** All query strings are included.
- **Cache compressed objects setting:** Enabled. For more information, see Compression support.

> **Warning**
>
> Because this policy has a minimum TTL greater than 0, CloudFront will cache content for at least the duration specified in the cache policy's minimum TTL, even if the `Cache-Control: no-cache`, `no-store`, or `private` directives are present in the origin headers.

AWS Amplify Hosting cache policies

Amplify uses the following managed cache policies to optimize the default cache configuration for customers' applications:

- Amplify-Default
- Amplify-DefaultNoCookies
- Amplify-ImageOptimization
- Amplify-StaticContent

> **Note**
>
> These policies are only used by Amplify. We don't recommend that you use these policies for your distributions.

For more information about managing cache configuration for your Amplify hosted application, see Managing cache configuration in the *Amplify Hosting User Guide*.

CachingDisabled

View this policy in the CloudFront console

This policy disables caching. This policy is useful for dynamic content and for requests that are not cacheable.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`4135ea2d-6df8-44a3-9df3-4b5a84be39ad`

This policy has the following settings:

- **Minimum TTL:** 0 seconds
- **Maximum TTL:** 0 seconds
- **Default TTL:** 0 seconds
- **Headers included in the cache key:** None
- **Cookies included in the cache key:** None

- **Query strings included in the cache key:** None
- **Cache compressed objects setting:** Disabled

CachingOptimized

View this policy in the CloudFront console

This policy is designed to optimize cache efficiency by minimizing the values that CloudFront includes in the cache key. CloudFront doesn't include any query strings or cookies in the cache key, and only includes the normalized `Accept-Encoding` header. This enables CloudFront to separately cache objects in the Gzip and Brotli compressions formats when the origin returns them or when CloudFront edge compression is enabled.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`658327ea-f89d-4fab-a63d-7e88639e58f6`

This policy has the following settings:

- **Minimum TTL:** 1 second.
- **Maximum TTL:** 31,536,000 seconds (365 days).
- **Default TTL:** 86,400 seconds (24 hours).
- **Headers included in the cache key:** None are explicitly included. The normalized `Accept-Encoding` header is included because the cache compressed objects setting is enabled. For more information, see Compression support.
- **Cookies included in the cache key:** None.
- **Query strings included in the cache key:** None.
- **Cache compressed objects setting:** Enabled. For more information, see Compression support.

> **Warning**
>
> Because this policy has a minimum TTL greater than 0, CloudFront will cache content for at least the duration specified in the cache policy's minimum TTL, even if the `Cache-Control: no-cache`, `no-store`, or `private` directives are present in the origin headers.

CachingOptimizedForUncompressedObjects

View this policy in the CloudFront console

This policy is designed to optimize cache efficiency by minimizing the values included in the cache key. No query strings, headers, or cookies are included. This policy is identical to the previous one, but it disables the cache compressed objects setting.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`b2884449-e4de-46a7-ac36-70bc7f1ddd6d`

This policy has the following settings:

- **Minimum TTL:** 1 second
- **Maximum TTL:** 31,536,000 seconds (365 days)
- **Default TTL:** 86,400 seconds (24 hours)
- **Headers included in the cache key:** None
- **Cookies included in the cache key:** None
- **Query strings included in the cache key:** None
- **Cache compressed objects setting:** Disabled

> **Warning**
>
> Because this policy has a minimum TTL greater than 0, CloudFront will cache content for at least the duration specified in the cache policy's minimum TTL, even if the `Cache-Control: no-cache`, `no-store`, or `private` directives are present in the origin headers.

Elemental-MediaPackage

View this policy in the CloudFront console

This policy is designed for use with an origin that is an AWS Elemental MediaPackage endpoint.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`08627262-05a9-4f76-9ded-b50ca2e3a84f`

This policy has the following settings:

- **Minimum TTL:** 0 seconds
- **Maximum TTL:** 31,536,000 seconds (365 days)
- **Default TTL:** 86,400 seconds (24 hours)
- **Headers included in the cache key:**
  - `Origin`

  The normalized `Accept-Encoding` header is also included because the cache compressed objects setting is enabled for Gzip. For more information, see Compression support.
- **Cookies included in the cache key:** None
- **Query strings included in the cache key:**
  - `aws.manifestfilter`
  - `start`
  - `end`
  - `m`
- **Cache compressed objects setting:** Enabled for Gzip. For more information, see Compression support.

UseOriginCacheControlHeaders

View this policy in the CloudFront console

This policy is designed for use with an origin that returns `Cache-Control` HTTP response headers and does not serve different content based on values present in the query string. If your origin serves different content based on values present in the query string, consider using UseOriginCacheControlHeaders-QueryStrings.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`83da9c7e-98b4-4e11-a168-04f0df8e2c65`

This policy has the following settings:

- **Minimum TTL:** 0 seconds

- **Maximum TTL:** 31,536,000 seconds (365 days)
- **Default TTL:** 0 seconds
- **Headers included in the cache key:**
  - `Host`
  - `Origin`
  - `X-HTTP-Method-Override`
  - `X-HTTP-Method`
  - `X-Method-Override`

  The normalized `Accept-Encoding` header is also included because the cache compressed objects setting is enabled. For more information, see Compression support.
- **Cookies included in the cache key:** All cookies are included.
- **Query strings included in the cache key:** None.
- **Cache compressed objects setting:** Enabled. For more information, see Compression support.

UseOriginCacheControlHeaders-QueryStrings

View this policy in the CloudFront console

This policy is designed for use with an origin that returns `Cache-Control` HTTP response headers and serves different content based on values present in the query string. If your origin does not serve different content based on values present in the query string, consider using UseOriginCacheControlHeaders.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`4cc15a8a-d715-48a4-82b8-cc0b614638fe`

This policy has the following settings:

- **Minimum TTL:** 0 seconds
- **Maximum TTL:** 31,536,000 seconds (365 days)
- **Default TTL:** 0 seconds
- **Headers included in the cache key:**
  - `Host`

- `Origin`
- `X-HTTP-Method-Override`
- `X-HTTP-Method`
- `X-Method-Override`

The normalized `Accept-Encoding` header is also included because the cache compressed objects setting is enabled. For more information, see Compression support.

- **Cookies included in the cache key:** All cookies are included.
- **Query strings included in the cache key:** All query strings are included.
- **Cache compressed objects setting:** Enabled. For more information, see Compression support.

Understand the cache key

The *cache key* determines whether a viewer request to a CloudFront edge location results in a *cache hit*. The cache key is the unique identifier for an object in the cache. Each object in the cache has a unique cache key.

A cache hit occurs when a viewer request generates the same cache key as a prior request, and the object for that cache key is in the edge location's cache and valid. When there's a cache hit, the requested object is served to the viewer from a CloudFront edge location, which has the following benefits:

- Reduced load on your origin server
- Reduced latency for the viewer

You can get better performance from your website or application when you have a higher *cache hit ratio* (a higher proportion of viewer requests that result in a cache hit). One way to improve your cache hit ratio is to include only the minimum necessary values in the cache key. For more information, see the following sections.

You can modify the values (URL query strings, HTTP headers, and cookies) in the cache key by using a cache policy. (You can also modify the cache key by using a Lambda@Edge function or a CloudFront Function on a viewer request.) Before modifying the cache key, it's important to understand how your application is designed and when and how it might serve different responses based on characteristics of the viewer request. When a value in the viewer request determines the response that your origin returns, you should include that value in the cache key. But if you include

a value in the cache key that doesn't affect the response that your origin returns, you might end up caching duplicate objects.

Default cache key

By default, the cache key for a CloudFront distribution includes the following information:

- The domain name of the CloudFront distribution (for example, d111111abcdef8.cloudfront.net)
- The URL path of the requested object (for example, `/content/stories/example-story.html`)

> **Note**
>
> The `OPTIONS` method is included in the cache key for `OPTIONS` requests. This means that responses to `OPTIONS` requests are cached separately from responses to `GET` and `HEAD` requests.

Other values from the viewer request are not included in the cache key, by default. Consider the following HTTP request from a web browser.

```
            GET /content/stories/example-story.html?ref=0123abc&split-pages=false
HTTP/1.1
            Host: d111111abcdef8.cloudfront.net
            User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/68.0
            Accept: text/html,*/*
            Accept-Language: en-US,en
            Cookie: session_id=01234abcd
            Referer: https://news.example.com/
```

When a viewer request like this example comes in to a CloudFront edge location, CloudFront uses the cache key to determine if there's a cache hit. By default, only the following components of the request are included in the cache key: `/content/stories/example-story.html` and `d111111abcdef8.cloudfront.net`. If the requested object is not in the cache (a cache miss), then CloudFront sends a request to the origin to get the object. After getting the object, CloudFront returns it to the viewer and stores it in the edge location's cache.

When CloudFront receives another request for the same object, as determined by the cache key, CloudFront serves the cached object to the viewer immediately, without sending a request to the origin. For example, consider the following HTTP request that comes in after the previous request.

```
            GET /content/stories/example-story.html?ref=xyz987&split-pages=true
HTTP/1.1
            Host: d111111abcdef8.cloudfront.net
            User-Agent: Mozilla/5.0 AppleWebKit/537.36 Chrome/83.0.4103.116
            Accept: text/html,*/*
            Accept-Language: en-US,en
            Cookie: session_id=wxyz9876
            Referer: https://rss.news.example.net/
```

This request is for the same object as the previous request, but is different from the previous request. It has a different URL query string, different `User-Agent` and `Referer` headers, and a different `session_id` cookie. However, none of these values are part of the cache key by default, so this second request results in a cache hit.

Customize the cache key

In some cases, you might want to include more information in the cache key, even though doing so might result in fewer cache hits. You specify what to include in the cache key by using a cache policy.

For example, if your origin server uses the `Accept-Language` HTTP header in viewer requests to return different content based on the viewer's language, you might want to include this header in the cache key. When you do that, CloudFront uses this header to determine cache hits, and includes the header in *origin requests* (requests that CloudFront sends to the origin when there's a cache miss).

One potential consequence of including additional values in the cache key is that CloudFront might end up caching duplicate objects because of the variation that can occur in viewer requests. For example, viewers might send any of the following values for the `Accept-Language` header:

- `en-US,en`
- `en,en-US`
- `en-US, en`

- `en-US`

All of these different values indicate that the viewer's language is English, but the variation can cause CloudFront to cache the same object multiple times. This can reduce cache hits and increase the number of origin requests. You could avoid this duplication by not including the `Accept-Language` header in the cache key, and instead configuring your website or application to use different URLs for content in different languages (for example, `/en-US/content/stories/example-story.html`).

For any given value that you intend to include in the cache key, you should make sure that you understand how many different variations of that value might appear in viewer requests. For certain request values, it rarely makes sense to include them in the cache key. For example, the `User-Agent` header can have thousands of unique variations, so it's generally not a good candidate for including in the cache key. Cookies that have user-specific or session-specific values and are unique across thousands (or even millions) of requests are also not good candidates for cache key inclusion. If you do include these values in the cache key, each unique variation results in another copy of the object in the cache. If these copies of the object are not unique, or if you end up with such a large number of slightly different objects that each object only gets a small number of cache hits, you might want to consider a different approach. You can exclude these highly variable values from the cache key, or you can mark objects as non-cacheable.

Use caution when customizing the cache key. Sometimes it's desirable, but it can have unintended consequences such as caching duplicate objects, lowering your cache hit ratio, and increasing the number of origin requests. Sometimes your origin website or application needs certain values from viewer requests for analytics, telemetry, or other uses, but those values don't change the object that the origin returns. In this case, use an origin request policy to include these values in origin requests but *not* include them in the cache key.

Control origin requests with a policy

When a viewer request to CloudFront results in a *cache miss* (the requested object is not cached at the edge location), CloudFront sends a request to the origin to retrieve the object. This is called an *origin request*. The origin request always includes the following information from the viewer request:

- The URL path (the path only, without URL query strings or the domain name)
- The request body (if there is one)
- The HTTP headers that CloudFront automatically includes in every origin request, including `Host`, `User-Agent`, and `X-Amz-Cf-Id`

Other information from the viewer request, such as URL query strings, HTTP headers, and cookies, is not included in the origin request by default. (Exception: With legacy cache settings, CloudFront forwards the headers to your origin by default.) However, you might want to receive some of this other information at the origin, for example to collect data for analytics or telemetry. You can use an *origin request policy* to control the information that's included in an origin request.

Origin request policies are separate from cache policies, which control the cache key. This way, you can receive additional information at the origin and also maintain a good *cache hit ratio* (the proportion of viewer requests that result in a cache hit). You do this by separately controlling which information is included in origin requests (using the origin request policy) and which is included in the cache key (using the cache policy).

Although the two kinds of policies are separate, they are related. All URL query strings, HTTP headers, and cookies that you include in the cache key (using a cache policy) are automatically included in origin requests. Use the origin request policy to specify the information that you want to include in origin requests, but *not* include in the cache key. Just like a cache policy, you attach an origin request policy to one or more cache behaviors in a CloudFront distribution.

You can also use an origin request policy to add additional HTTP headers to an origin request that were not included in the viewer request. These additional headers are added by CloudFront before sending the origin request, with header values that are determined automatically based on the viewer request. For more information, see the section called “Add CloudFront request headers”.

Topics

- Understand origin request policies

- Create origin request policies
- Use managed origin request policies
- Add CloudFront request headers
- Understand how origin request policies and cache policies work together

Understand origin request policies

CloudFront provides some predefined origin request policies, known as *managed policies*, for common use cases. You can use these managed policies, or you can create your own origin request policy that's specific to your needs. For more information about the managed policies, see Use managed origin request policies.

An origin request policy contains the following settings, which are categorized into *policy information* and *origin request settings*.

Policy information

Name

A name to identify the origin request policy. In the console, you use the name to attach the origin request policy to a cache behavior.

Description

A comment to describe the origin request policy. This is optional.

Origin request settings

Origin request settings specify the values in viewer requests that are included in requests that CloudFront sends to the origin (known as origin requests). The values can include URL query strings, HTTP headers, and cookies. The values that you specify are included in origin requests, but are not included in the cache key. For information about controlling the cache key, see Control the cache key with a policy.

Headers

The HTTP headers in viewer requests that CloudFront includes in origin requests. For headers, you can choose one of the following settings:

- **None** – The HTTP headers in viewer requests are *not* included in origin requests.

- **All viewer headers** – All HTTP headers in viewer requests are included in origin requests.
- **All viewer headers and the following CloudFront headers** – All HTTP headers in viewer requests are included in origin requests. Additionally, you specify which of the CloudFront headers you want to add to origin requests. For more information about the CloudFront headers, see the section called “Add CloudFront request headers”.
- **Include the following headers** – You specify which HTTP headers are included in origin requests.

  > **Note**
  >
  > Do not specify a header that is already included in your **Origin Custom Headers** settings. For more information, see Configure CloudFront to add custom headers to origin requests.

- **All viewer headers except** – You specify which HTTP headers are ***not*** included in origin requests. All other HTTP headers in viewer requests, except for the ones specified, are included.

When you use the **All viewer headers and the following CloudFront headers**, **Include the following headers**, or **All viewer headers except** setting, you specify HTTP headers by the header name only. CloudFront includes the full header, including its value, in origin requests.

> **Note**
>
> When you use the **All viewer headers except** setting to remove the viewer's `Host` header, CloudFront adds a new `Host` header with the origin's domain name to the origin request.

Cookies

The cookies in viewer requests that CloudFront includes in origin requests. For cookies, you can choose one of the following settings:

- **None** – The cookies in viewer requests are *not* included in origin requests.
- **All** – All cookies in viewer requests are included in origin requests.
- **Include the following cookies** – You specify which cookies in viewer requests are included in origin requests.

- **All cookies except** – You specify which cookies in viewer requests are ***not*** included in origin requests. All other cookies in viewer requests are included.

When you use the **Include the following cookies** or **All cookies except** setting, you specify cookies by their name only. CloudFront includes the full cookie, including its value, in origin requests.

You can use the asterisk (*) as a wildcard in cookie names. It matches 0 or more characters in the cookie name.

For example, if you specify `userid_*`, CloudFront matches all of the cookies whose names begin with `userid_`, such as `userid_1234` and `userid_5678`. A cookie name can't consist only of asterisks. To include all cookies, use the **All** setting instead.

Query strings

The URL query strings in viewer requests that CloudFront includes in origin requests. For query strings, you can choose one of the following settings:

- **None** – The query strings in viewer requests are *not* included in origin requests.
- **All** – All query strings in viewer requests are included in origin requests.
- **Include the following query strings** – You specify which query strings in viewer requests are included in origin requests.
- **All query strings except** – You specify which query strings in viewer requests are ***not*** included in origin requests. All other query strings are included.

When you use the **Include the following query strings** or **All query strings except** setting, you specify query strings by their name only. CloudFront includes the full query string, including its value, in origin requests.

Create origin request policies

You can use an origin request policy to control the values (URL query strings, HTTP headers, and cookies) that are included in requests that CloudFront sends to your origin. You can create an origin request policy in the CloudFront console, with the AWS Command Line Interface (AWS CLI), or with the CloudFront API.

After you create an origin request policy, you attach it to one or more cache behaviors in a CloudFront distribution.

Origin request policies are not required. When a cache behavior does not have an origin request policy attached, the origin request includes all the values that are specified in the cache policy, but nothing more.

> **Note**
>
> To use an origin request policy, the cache behavior must also use a cache policy. You cannot use an origin request policy in a cache behavior without a cache policy.

> **Note**
>
> If you create a custom origin request policy for an Amazon S3 origin, we recommend that you don't include `response-content-type`, `response-content-disposition`, or `response-content-encoding`. If your policy includes all query strings, use an allow list instead.

Console

To create an origin request policy (console)

1. Sign in to the AWS Management Console and open the **Policies** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home?#/policies.
2. Choose **Origin request**, then choose **Create origin request policy**.
3. Choose the desired setting for this origin request policy. For more information, see Understand origin request policies.
4. When finished, choose **Create**.

After you create an origin request policy, you can attach it to a cache behavior.

To attach an origin request policy to an existing distribution (console)

1. Open the **Distributions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/distributions.
2. Choose the distribution to update, then choose the **Behaviors** tab.

3. Choose the cache behavior to update, then choose **Edit**.

   Or, to create a new cache behavior, choose **Create behavior**.
4. In the **Cache key and origin requests** section, make sure that **Cache policy and origin request policy** is chosen.
5. For **Origin request policy**, choose the origin request policy to attach to this cache behavior.
6. At the bottom of the page, choose **Save changes**.

To attach an origin request policy to a new distribution (console)

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **Create distribution**.
3. In the **Cache key and origin requests** section, make sure that **Cache policy and origin request policy** is chosen.
4. For **Origin request policy**, choose the origin request policy to attach to this distribution's default cache behavior.
5. Choose the desired settings for the origin, default cache behavior, and other distribution settings. For more information, see All distribution settings reference.
6. When finished, choose **Create distribution**.

CLI

To create an origin request policy with the AWS Command Line Interface (AWS CLI), use the **aws cloudfront create-origin-request-policy** command. You can use an input file to provide the command's input parameters, rather than specifying each individual parameter as command line input.

To create an origin request policy (CLI with input file)

1. Use the following command to create a file named `origin-request-policy.yaml` that contains all of the input parameters for the **create-origin-request-policy** command.

```
aws cloudfront create-origin-request-policy --generate-cli-skeleton yaml-input >
 origin-request-policy.yaml
```

2. Open the file named `origin-request-policy.yaml` that you just created. Edit the file to specify the origin request policy settings that you want, then save the file. You can remove optional fields from the file, but don't remove the required fields.

   For more information about the origin request policy settings, see Understand origin request policies.

3. Use the following command to create the origin request policy using input parameters from the `origin-request-policy.yaml` file.

   ```
   aws cloudfront create-origin-request-policy --cli-input-yaml file://origin-
   request-policy.yaml
   ```

   Make note of the `Id` value in the command's output. This is the origin request policy ID, and you need it to attach the origin request policy to a CloudFront distribution's cache behavior.

To attach an origin request policy to an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to update. Replace *`distribution_ID`* with the distribution's ID.

   ```
   aws cloudfront get-distribution-config --id distribution_ID --output yaml >
    dist-config.yaml
   ```

2. Open the file named `dist-config.yaml` that you just created. Edit the file, making the following changes to each cache behavior that you are updating to use an origin request policy.

   - In the cache behavior, add a field named `OriginRequestPolicyId`. For the field's value, use the origin request policy ID that you noted after creating the policy.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the distribution to use the origin request policy. Replace *`distribution_ID`* with the distribution's ID.

```
aws cloudfront update-distribution --id distribution_ID --cli-input-yaml file://
dist-config.yaml
```

To attach an origin request policy to a new distribution (CLI with input file)

1. Use the following command to create a file named `distribution.yaml` that contains all of the input parameters for the **create-distribution** command.

   ```
   aws cloudfront create-distribution --generate-cli-skeleton yaml-input >
    distribution.yaml
   ```

2. Open the file named `distribution.yaml` that you just created. In the default cache behavior, in the `OriginRequestPolicyId` field, enter the origin request policy ID that you noted after creating the policy. Continue editing the file to specify the distribution settings that you want, then save the file when finished.

   For more information about the distribution settings, see All distribution settings reference.

3. Use the following command to create the distribution using input parameters from the `distribution.yaml` file.

   ```
   aws cloudfront create-distribution --cli-input-yaml file://distribution.yaml
   ```

API

To create an origin request policy with the CloudFront API, use CreateOriginRequestPolicy. For more information about the fields that you specify in this API call, see Understand origin request policies and the API reference documentation for your AWS SDK or other API client.

After you create an origin request policy, you can attach it to a cache behavior, using one of the following API calls:

- To attach it to a cache behavior in an existing distribution, use UpdateDistribution.
- To attach it to a cache behavior in a new distribution, use CreateDistribution.

For both of these API calls, provide the origin request policy's ID in the `OriginRequestPolicyId` field, inside a cache behavior. For more information about the other fields that you specify in these API calls, see All distribution settings reference and the API reference documentation for your AWS SDK or other API client.

Use managed origin request policies

CloudFront provides a set of managed origin request policies that you can attach to any of your distribution's cache behaviors. With a managed origin request policy, you don't need to write or maintain your own origin request policy. The managed policies use settings that are optimized for specific use cases.

Important

When you use the `AllViewer`, `AllViewerAndCloudFrontHeaders-2022-06`, `AllViewerExceptHostHeader`, or `Elemental-MediaTailor-PersonalizedManifests` managed origin request policy with an Amazon S3 origin, CloudFront doesn't forward the following query string parameters to the origin:

- `response-content-type`
- `response-content-disposition`
- `response-content-encoding`

These parameters change the response headers that Amazon S3 returns. CloudFront continues to forward all other query strings that the policy includes.
This applies only to managed origin request policies. If you use a custom origin request policy, CloudFront forwards these parameters as configured. If your application requires these parameters to be forwarded to an Amazon S3 origin, use a custom origin request policy, or contact Support. These parameters have no effect on non-Amazon S3 origins.

To use a managed origin request policy, you attach it to a cache behavior in your distribution. The process is the same as when you create an origin request policy, but instead of creating a new one, you just attach one of the managed origin request policies. You attach the policy either by name (with the console) or by ID (with the AWS CLI or SDKs). The names and IDs are listed in the following section.

For more information, see Create origin request policies.

The following topics describe the managed origin request policies that you can use.

Topics

- AllViewer
- AllViewerAndCloudFrontHeaders-2022-06
- AllViewerExceptHostHeader
- CORS-CustomOrigin
- CORS-S3Origin
- Elemental-MediaTailor-PersonalizedManifests
- HostHeaderOnly
- UserAgentRefererHeaders

AllViewer

View this policy in the CloudFront console

This policy includes all values (headers, cookies, and query strings) from the viewer request.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`216adef6-5c7f-47e4-b989-5492eafa07d3`

This policy has the following settings:

- **Headers included in origin requests:** All headers in the viewer request
- **Cookies included in origin requests:** All
- **Query strings included in origin requests:** All. For Amazon S3 origins, CloudFront doesn't forward `response-content-type`, `response-content-disposition`, or `response-content-encoding`. For more information, see the important note at the beginning of this topic.

AllViewerAndCloudFrontHeaders-2022-06

View this policy in the CloudFront console

This policy includes all values (headers, cookies, and query strings) from the viewer request, and all CloudFront headers that were released through June 2022 (CloudFront headers released after June 2022 are not included).

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`33f36d7e-f396-46d9-90e0-52428a34d9dc`

This policy has the following settings:

- **Headers included in origin requests:** All headers in the viewer request, and the following CloudFront headers:
  - `CloudFront-Forwarded-Proto`
  - `CloudFront-Is-Android-Viewer`
  - `CloudFront-Is-Desktop-Viewer`
  - `CloudFront-Is-IOS-Viewer`
  - `CloudFront-Is-Mobile-Viewer`
  - `CloudFront-Is-SmartTV-Viewer`
  - `CloudFront-Is-Tablet-Viewer`
  - `CloudFront-Viewer-Address`
  - `CloudFront-Viewer-ASN`
  - `CloudFront-Viewer-City`
  - `CloudFront-Viewer-Country`
  - `CloudFront-Viewer-Country-Name`
  - `CloudFront-Viewer-Country-Region`
  - `CloudFront-Viewer-Country-Region-Name`
  - `CloudFront-Viewer-Http-Version`
  - `CloudFront-Viewer-Latitude`
  - `CloudFront-Viewer-Longitude`
  - `CloudFront-Viewer-Metro-Code`
  - `CloudFront-Viewer-Postal-Code`
  - `CloudFront-Viewer-Time-Zone`
  - `CloudFront-Viewer-TLS`
- **Cookies included in origin requests:** All

- **Query strings included in origin requests:** All. For Amazon S3 origins, CloudFront doesn't forward `response-content-type`, `response-content-disposition`, or `response-content-encoding`. For more information, see the important note at the beginning of this topic.

AllViewerExceptHostHeader

View this policy in the CloudFront console

This policy does ***not*** include the `Host` header from the viewer request, but does include all other values (headers, cookies, and query strings) from the viewer request.

This policy also includes additional CloudFront request headers for HTTP protocol, HTTP version, TLS version, and all device type and viewer location headers.

This policy is intended for use with Amazon API Gateway, AWS Lambda function URL, and Lambda Web Function origins. These origins expect the `Host` header to contain the origin domain name, not the domain name of the CloudFront distribution. Forwarding the `Host` header from the viewer request to these origins can prevent them from working.

> **Note**
>
> When you use this managed origin request policy to remove the viewer's `Host` header, CloudFront adds a new `Host` header with the origin's domain name to the origin request.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`b689b0a8-53d0-40ab-baf2-68738e2966ac`

This policy has the following settings:

- **Headers included in origin requests:** All headers in the viewer request ***except*** for the `Host` header
- **Cookies included in origin requests:** All
- **Query strings included in origin requests:** All. For Amazon S3 origins, CloudFront doesn't forward `response-content-type`, `response-content-disposition`, or `response-content-encoding`. For more information, see the important note at the beginning of this topic.

CORS-CustomOrigin

View this policy in the CloudFront console

This policy includes the header that enables cross-origin resource sharing (CORS) requests when the origin is a custom origin.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`59781a5b-3903-41f3-afcb-af62929ccde1`

This policy has the following settings:

- **Headers included in origin requests:**
  - `Origin`
- **Cookies included in origin requests:** None
- **Query strings included in origin requests:** None

CORS-S3Origin

View this policy in the CloudFront console

This policy includes the headers that enable cross-origin resource sharing (CORS) requests when the origin is an Amazon S3 bucket.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`88a5eaf4-2fd4-4709-b370-b4c650ea3fcf`

This policy has the following settings:

- **Headers included in origin requests:**
  - `Origin`
  - `Access-Control-Request-Headers`
  - `Access-Control-Request-Method`
- **Cookies included in origin requests:** None
- **Query strings included in origin requests:** None

Elemental-MediaTailor-PersonalizedManifests

View this policy in the CloudFront console

This policy is intended for use with an origin that is an AWS Elemental MediaTailor endpoint.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`775133bc-15f2-49f9-abea-afb2e0bf67d2`

This policy has the following settings:

- **Headers included in origin requests:**
  - `Origin`
  - `Access-Control-Request-Headers`
  - `Access-Control-Request-Method`
  - `User-Agent`
  - `X-Forwarded-For`
- **Cookies included in origin requests:** None
- **Query strings included in origin requests:** All. For Amazon S3 origins, CloudFront doesn't forward `response-content-type`, `response-content-disposition`, or `response-content-encoding`. For more information, see the important note at the beginning of this topic.

HostHeaderOnly

View this policy in the CloudFront console

This policy includes only the `Host` header from the origin request. It doesn't include any query strings or cookies.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`bf0718e1-ba1e-49d1-88b1-f726733018ae`

This policy has the following settings:

- **Headers included in origin requests:** Host

- **Cookies included in origin requests:** None
- **Query strings included in origin requests:** None

UserAgentRefererHeaders

View this policy in the CloudFront console

This policy includes only the `User-Agent` and `Referer` headers. It doesn't include any query strings or cookies.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`acba4595-bd28-49b8-b9fe-13317c0390fa`

This policy has the following settings:

- **Headers included in origin requests:**
  - `User-Agent`
  - `Referer`
- **Cookies included in origin requests:** None
- **Query strings included in origin requests:** None

Add CloudFront request headers

You can configure CloudFront to add specific HTTP headers to the requests that CloudFront receives from viewers and forwards on to your origin or edge function. The values of these HTTP headers are based on characteristics of the viewer or the viewer request. The headers provide information about the viewer's device type, IP address, geographic location, request protocol (HTTP or HTTPS), HTTP version, TLS connection details, JA3 fingerprint, and JA4 fingerprint. You can also configure your distribution's cache behavior to forward WebSocket headers. For more information, see Use WebSockets with CloudFront distributions.

With these headers, your origin or your edge function can receive information about the viewer without the need for you to write your own code to determine this information. If your origin returns different responses based on the information in these headers, you can include them in the *cache key* so that CloudFront caches the responses separately. For example, your origin might respond with content in a specific language based on the country that the viewer is in, or with

content tailored to a specific device type. Your origin might also write these headers to log files, which you can use to determine information about where your viewers are, which device types they're on, and more.

To include these headers in the cache key, use a *cache policy*. For more information, see *Control the cache key with a policy* and the section called "Understand the cache key".

To receive these headers at your origin but not include them in the cache key, use an *origin request policy*. For more information, see *Control origin requests with a policy*.

Topics

- Device type headers
- Viewer location headers
- Headers to determine the viewer's header structure
- TLS-related headers
- Other CloudFront headers

Device type headers

You can add the following headers to determine the viewer's device type. Based on the value of the `User-Agent` header, CloudFront sets the value of these headers to `true` or `false`. If a device falls into more than one category, more than one value can be `true`. For example, for some tablet devices, CloudFront sets both `CloudFront-Is-Mobile-Viewer` and `CloudFront-Is-Tablet-Viewer` to `true`.

- `CloudFront-Is-Android-Viewer` – Set to `true` when CloudFront determines that the viewer is a device with the Android operating system.
- `CloudFront-Is-Desktop-Viewer` – Set to `true` when CloudFront determines that the viewer is a desktop device.
- `CloudFront-Is-IOS-Viewer` – Set to `true` when CloudFront determines that the viewer is a device with an Apple mobile operating system, such as iPhone, iPod touch, and some iPad devices.
- `CloudFront-Is-Mobile-Viewer` – Set to `true` when CloudFront determines that the viewer is a mobile device.
- `CloudFront-Is-SmartTV-Viewer` – Set to `true` when CloudFront determines that the viewer is a smart TV.

- `CloudFront-Is-Tablet-Viewer` – Set to `true` when CloudFront determines that the viewer is a tablet.

Viewer location headers

You can add the following headers to determine the viewer's location. CloudFront determines the values for these headers based on the viewer's IP address. For non-ASCII characters in these headers' values, CloudFront percent-encodes the character according to section 1.2 of RFC 3986.

- `CloudFront-Viewer-Address` – Contains the IP address of the viewer and the source port of the request. For example, a header value of `198.51.100.10:46532` means the viewer's IP address is 198.51.100.10 and the request source port is 46532.
- `CloudFront-Viewer-ASN` – Contains the autonomous system number (ASN) of the viewer.

> **Note**
>
> `CloudFront-Viewer-Address` and `CloudFront-Viewer-ASN` can be added in an origin request policy, but not in a cache policy.

- `CloudFront-Viewer-ISP` – Contains the name of the viewer's internet service provider (ISP). You can add this header only in an origin request policy, not in a cache policy or with legacy cache settings. CloudFront doesn't percent-encode this header's value, and doesn't add the header when it can't determine the viewer's ISP.
- `CloudFront-Viewer-Country` – Contains the two-letter country code for the viewer's country. For a list of country codes, see ISO 3166-1 alpha-2.
- `CloudFront-Viewer-City` – Contains the name of the viewer's city.

When you add the following headers, CloudFront applies them to all requests *except* those that originate from the AWS network:

- `CloudFront-Viewer-Country-Name` – Contains the name of the viewer's country.
- `CloudFront-Viewer-Country-Region` – Contains a code (up to three characters) that represents the viewer's region. The region is the first-level subdivision (the broadest or least specific) of the ISO 3166-2 code.
- `CloudFront-Viewer-Country-Region-Name` – Contains the name of the viewer's region. The region is the first-level subdivision (the broadest or least specific) of the ISO 3166-2 code.

- `CloudFront-Viewer-Latitude` – Contains the viewer's approximate latitude.
- `CloudFront-Viewer-Longitude` – Contains the viewer's approximate longitude.
- `CloudFront-Viewer-Metro-Code` – Contains the viewer's metro code. This is present only when the viewer is in the United States.
- `CloudFront-Viewer-Postal-Code` – Contains the viewer's postal code.
- `CloudFront-Viewer-Time-Zone` – Contains the viewer's time zone, in IANA time zone database format (for example, `America/Los_Angeles`).

> **Note**
>
> `CloudFront-Viewer-City`, `CloudFront-Viewer-Metro-Code`, and `CloudFront-Viewer-Postal-Code` might not be available for every IP address. Some IP addresses can't be geolocated with enough specificity to get that information.

Headers to determine the viewer's header structure

You can add the following headers to help identify the viewer based on the headers that it sends. For example, different browsers may send HTTP headers in a certain order. If the browser specified in the `User-Agent` header doesn't match that browser's expected header order, you can deny the request. Additionally, if the `CloudFront-Viewer-Header-Count` value does not match the number of headers in `CloudFront-Viewer-Header-Order`, you can deny the request.

- `CloudFront-Viewer-Header-Order` – Contains the viewer's header names in the order requested, separated by a colon. For example: `CloudFront-Viewer-Header-Order: Host:User-Agent:Accept:Accept-Encoding`. Headers beyond the character limit of 7,680 are truncated.
- `CloudFront-Viewer-Header-Count` – Contains the total number of the viewer's headers.

TLS-related headers

You can add the following headers to determine the viewer's JA3 fingerprint, JA4 fingerprint, and TLS connection details. The TLS connection details include the key exchange and signature algorithms that the viewer and CloudFront negotiated.

- `CloudFront-Viewer-JA3-Fingerprint` – Contains the JA3 fingerprint of the viewer. The JA3 fingerprint can help you determine whether the request comes from a known client, whether that's malware or a malicious bot, or an expected (allow-listed) application.
- `CloudFront-Viewer-JA4-Fingerprint` – Contains the JA4 fingerprint of the viewer. Similar to the JA3 fingerprint, the JA4 fingerprint can help you determine whether the request comes from a known client, whether that's malware or a malicious bot, or an expected (allow-listed) application. You can use the fingerprint to build a database of known good and bad actors to apply when inspecting HTTP requests. You can then inspect the header value on your application web servers or in your Lambda@Edge and CloudFront Functions to compare the header value against a list of known malware fingerprints to block malicious clients.
- `CloudFront-Viewer-TLS` – Contains the SSL/TLS version, the cipher, and information about the SSL/TLS handshake that was used for the connection between the viewer and CloudFront. The header value is in the following format:

  ```
  SSL/TLS_version:cipher:handshake_information
  ```

  For *`handshake_information`*, the header can contain the following values:

  - `fullHandshake` – A full handshake was performed for the SSL/TLS session.
  - `sessionResumed` – A previous SSL/TLS session was resumed.
  - `connectionReused` – A previous SSL/TLS connection was reused.

  The following are some example values for this header:

  ```
  TLSv1.3:TLS_AES_128_GCM_SHA256:sessionResumed
  ```

  ```
  TLSv1.2:ECDHE-ECDSA-AES128-GCM-SHA256:connectionReused
  ```

  ```
  TLSv1.1:ECDHE-RSA-AES128-SHA256:fullHandshake
  ```

  ```
  TLSv1:ECDHE-RSA-AES256-SHA:fullHandshake
  ```

  For the full list of possible SSL/TLS versions and ciphers that can be in this header value, see the section called “Supported protocols and ciphers between viewers and CloudFront”.
- `CloudFront-Viewer-TLS-Details` – Contains the SSL/TLS version, the cipher, information about the SSL/TLS handshake, and the public-key algorithms that the viewer and CloudFront

negotiated for the connection. This header includes the same information as `CloudFront-Viewer-TLS`. It also adds the key exchange algorithm, the signature scheme, and the type of public key in the certificate that CloudFront presented to the viewer. You can use this header to determine whether a connection used a post-quantum key exchange. The header value is a JSON object in the following format:

```
{"tlsVersion":"SSL/
TLS_version","cipherSuite":"cipher","handshakeType":"handshake_information","keyExchange":"ke
```

The header value contains the following fields:

- `tlsVersion` – The SSL/TLS version that was used for the connection. For example, `TLSv1.3`.
- `cipherSuite` – The cipher that was used for the connection. For example, `TLS_AES_128_GCM_SHA256`.
- `handshakeType` – Information about the SSL/TLS handshake: `fullHandshake`, `sessionResumed`, or `connectionReused`. These are the same values as in the `CloudFront-Viewer-TLS` header.
- `keyExchange` – The key exchange group that was negotiated for the connection. For example, `x25519`, `secp256r1`, or `X25519MLKEM768`. For the list of classical and quantum-safe key exchanges that CloudFront supports, see the section called “OpenSSL, s2n, and RFC cipher names”.
- `signatureScheme` – The signature scheme that CloudFront used to authenticate itself to the viewer. For example, `ecdsa_secp256r1_sha256` or `rsa_pss_rsae_sha256`.
- `signaturePublicKeyType` – The type and size of the public key in the certificate that CloudFront presented to the viewer. For example, `rsa2048`, `ecdsa_secp256r1`, or `mldsa65`.

When a previous SSL/TLS session is resumed, CloudFront sets the fields that don't apply to the resumed session to `NONE`:

- For a resumed TLS 1.2 session, `keyExchange`, `signatureScheme`, and `signaturePublicKeyType` are `NONE`.
- For a resumed TLS 1.3 session, `keyExchange` contains the negotiated key exchange group because TLS 1.3 performs a new key exchange when it resumes a session. `signatureScheme` and `signaturePublicKeyType` are `NONE`.

If CloudFront can't determine the value of a field, the field is empty.

The following are some example values for this header:

```
{"tlsVersion":"TLSv1.3","cipherSuite":"TLS_AES_128_GCM_SHA256","handshakeType":"fullHandshake
```

```
{"tlsVersion":"TLSv1.3","cipherSuite":"TLS_AES_128_GCM_SHA256","handshakeType":"sessionResume
```

```
{"tlsVersion":"TLSv1.2","cipherSuite":"ECDHE-RSA-AES128-GCM-
SHA256","handshakeType":"fullHandshake","keyExchange":"secp256r1","signatureScheme":"rsa_pkcs
```

```
{"tlsVersion":"TLSv1.2","cipherSuite":"ECDHE-RSA-AES128-GCM-
SHA256","handshakeType":"sessionResumed","keyExchange":"NONE","signatureScheme":"NONE","signa
```

For the full list of possible SSL/TLS versions and ciphers that can be in this header value, see the section called "Supported protocols and ciphers between viewers and CloudFront".

Notes

- The JA3 and JA4 fingerprints are derived from the SSL/TLS `Client Hello` packet. They are only present for HTTPS requests.
- For these TLS-related headers, you can add them to an origin request policy, but not in a cache policy.

Other CloudFront headers

You can add the following headers to determine the viewer's original request URI, original request query string parameters and values, protocol, and version:

- `CloudFront-Error-Uri` – Contains the original request URI that was received from the viewer.
- `CloudFront-Error-Args` – Contains the original request query string parameters and values.
- `CloudFront-Forwarded-Proto` – Contains the protocol of the viewer's request (HTTP or HTTPS).
- `CloudFront-Viewer-Http-Version` – Contains the HTTP version of the viewer's request.

Understand how origin request policies and cache policies work together

You can use a CloudFront origin request policy to control the requests that CloudFront sends to the origin, which are called *origin requests*. To use an origin request policy, you must attach a cache policy to the same cache behavior. You cannot use an origin request policy in a cache behavior without a cache policy. For more information, see *Control origin requests with a policy*.

Origin request policies and cache policies work together to determine the values that CloudFront includes in origin requests. All URL query strings, HTTP headers, and cookies that you specify in the cache key (using a cache policy) are automatically included in origin requests. Any additional query strings, headers, and cookies that you specify in an origin request policy are also included in origin requests (but not in the cache key).

Origin request policies and cache policies have settings that might appear to conflict with each other. For example, one policy might allow certain values while another policy blocks them. The following table explains which values CloudFront includes in origin requests when you use the settings of an origin request policy and a cache policy together. These settings generally apply to all types of values: query strings, headers, and cookies. However, in a cache policy you cannot specify all headers or use a header block list.

	Origin request policy			
	None	**All**	**Allow list**	**Block list**
Cache policy				
None	No values from the viewer request are included in the origin request, except for the defaults that are included in every origin request. For more informati	All values from the viewer request are included in the origin request.	Only the values specified in the origin request policy are included in the origin request.	All values from the viewer request ***except*** those specified in the origin request policy are included in the origin request.

	Origin request policy			
	None	**All**	**Allow list**	**Block list**
	on, see *Control origin requests with a policy*.			
All **Note:** You cannot specify all headers in a cache policy.	All query strings and cookies from the viewer request are included in the origin request.	All values from the viewer request are included in the origin request.	All query strings and cookies from the viewer request, and any headers specified in the origin request policy, are included in the origin request.	All query strings and cookies from the viewer request are included in the origin request, even those specified in the origin request policy block list. The cache policy setting overrides the origin request policy block list.

	Origin request policy			
	None	**All**	**Allow list**	**Block list**
Allow list	Only the specified values from the viewer request are included in the origin request.	All values from the viewer request are included in the origin request.	All values specified in the cache policy or the origin request policy are included in the origin request.	The values specified in the cache policy are included in the origin request, even if those same values are specified in the origin request policy block list. The cache policy allow list overrides the origin request policy block list.
Block list **Note:** You cannot specify headers in a cache policy block list.	All query strings and cookies from the viewer request ***except*** those specified are included in the origin request.	All values from the viewer request are included in the origin request.	The values specified in the origin request policy are included in the origin request, even if those same values are specified in the cache policy block list. The origin request policy allow list overrides the cache policy block list.	All values from the viewer request ***except*** those specified in the cache policy or the origin request policy are included in the origin request.

Add or remove HTTP headers in CloudFront responses with a policy

You can configure CloudFront to modify the HTTP headers in the responses that it sends to viewers (web browsers and other clients). CloudFront can remove headers that it received from the origin, or add headers to the response, before sending the response to viewers. Making these changes doesn't require writing code or changing the origin.

For example, you can remove headers such as `X-Powered-By` and `Vary` so that CloudFront doesn't include these headers in the responses that it sends to viewers. Or, you can add HTTP headers such as the following:

- A `Cache-Control` header to control browser caching.
- An `Access-Control-Allow-Origin` header to enable cross-origin resource sharing (CORS). You can also add other CORS headers.
- A set of common security headers, such as `Strict-Transport-Security`, `Content-Security-Policy`, and `X-Frame-Options`.
- A `Server-Timing` header to see information that's related to the performance and routing of both the request and response through CloudFront.

To specify the headers that CloudFront adds or removes in HTTP responses, you use a *response headers policy*. You attach a response headers policy to one or more *cache behaviors*. CloudFront modifies the headers in the responses that it sends to requests that match the cache behavior. CloudFront modifies the headers in the responses that it serves from the cache and the ones that it forwards from the origin. If the origin response includes one or more of the headers that are added in a response headers policy, the policy can specify if CloudFront uses the header it received from the origin or overwrites that header with the one in the response headers policy.

Note

If you add headers that control browser caching to your response headers policies, such as `Cache-Control`, CloudFront adds these headers to the *viewer response only*. These headers don't affect how CloudFront caches the requested object.

CloudFront provides predefined response headers policies, known as *managed policies*, for common use cases. You can use these managed policies or create your own policies. You can attach a single response headers policy to multiple cache behaviors in multiple distributions in your AWS account.

For more information, see the following:

Topics

- Understand response headers policies
- Create response headers policies
- Use managed response headers policies

Understand response headers policies

You can use a response headers policy to specify the HTTP headers that Amazon CloudFront removes or adds in responses that it sends to viewers. For more information about response headers policies and reasons to use them, see *Add or remove response headers with a policy*.

The following topics explain the settings in a response headers policy. The settings are grouped into categories, which are represented in the following topics.

Topics

- Policy details (metadata)
- CORS headers
- Security headers
- Custom headers
- Remove headers
- Server-Timing header

Policy details (metadata)

The policy details settings contain metadata about a response headers policy.

- **Name** – A name to identify the response headers policy. In the console, you use the name to attach the policy to a cache behavior.
- **Description** (optional) – A comment to describe the response headers policy. This is optional, but it can help you identify the purpose of the policy.

CORS headers

The cross-origin resource sharing (CORS) settings allow you to add and configure CORS headers in a response headers policy.

This list focuses on how to specify settings and valid values in a response headers policy. For more information about each of these headers and how they're used for real-world CORS requests and responses, see cross-origin resource sharing in the MDN Web Docs and the CORS protocol spec.

Access-Control-Allow-Credentials

This is a Boolean setting (`true` or `false`) that determines if CloudFront adds the `Access-Control-Allow-Credentials` header in responses to CORS requests. When this setting is set to `true`, CloudFront adds the `Access-Control-Allow-Credentials: true` header in responses to CORS requests. Otherwise, CloudFront doesn't add this header to responses.

Access-Control-Allow-Headers

Specifies the header names that CloudFront uses as values for the `Access-Control-Allow-Headers` header in responses to CORS preflight requests. Valid values for this setting include HTTP header names or the wildcard character (*), which indicates that all headers are allowed.

> **Note**
>
> The `Authorization` header can't use a wildcard and must be listed explicitly.

Examples of valid use of the wildcard character

Example	Will match	Will not match
`x-amz-*`	`x-amz-test` `x-amz-`	`x-amz`
`x-*-amz`	`x-test-amz` `x--amz`	
`*`	All headers except `Authorization`	`Authorization`

Access-Control-Allow-Methods

Specifies the HTTP methods that CloudFront uses as values for the `Access-Control-Allow-Methods` header in responses to CORS preflight requests. Valid values are `GET`, `DELETE`, `HEAD`, `OPTIONS`, `PATCH`, `POST`, `PUT`, and `ALL`. `ALL` is a special value that includes all of the listed HTTP methods.

Access-Control-Allow-Origin

Specifies the values that CloudFront can use in the `Access-Control-Allow-Origin` response header. Valid values for this setting include a specific origin (such as `http://www.example.com`) or the wildcard character (*), which indicates that all origins are allowed.

Notes

- The wildcard character (*) is allowed as the leftmost subdomain (`*.example.org`).
- The wildcard character (*) *isn't* allowed in the following positions:
  - Top-level domains (`example.*`)
  - To the right of subdomains (`test.*.example.org`) or within any subdomains (`*test.example.org`)
  - Inside of terms (`exa*mple.org`)

For examples of ways to use the wildcard character, see the following table.

Example	Will match	Will not match

`http://*.example.org`	`http://www.example.org`
	
	`http://test.example.org`
		`https``://test.example.org`
		
		`https``://test.example.org:123`
		
		`http://test.example.org:123`

`*.example.org`	`test.example.org`	`http://test.example.org:123`

Example	Will match	Will not match
	test.test.example.org .example.org http://test.example.org https://test.example.org	https://test.example.org:123
example.org	http://example.org https://example.org	
http://example.org		**https**://example.org http://example.org:123
http://example.org:*	http://example.org:123 http://example.org	
http://example.org:1*3	http://example.org:123 http://example.org:1893 http://example.org:13	
.example.org:1	test.example.org:123	

Access-Control-Expose-Headers

Specifies the header names that CloudFront uses as values for the `Access-Control-Expose-Headers` header in responses to CORS requests. Valid values for this setting include HTTP header names or the wildcard character (*).

Access-Control-Max-Age

A number of seconds, which CloudFront uses as the value for the `Access-Control-Max-Age` header in responses to CORS preflight requests.

Origin override

A Boolean setting that determines how CloudFront behaves when the response from the origin contains one of the CORS headers that's also in the policy.

- When set to `true` and the origin response contains a CORS header that's also in the policy, CloudFront adds the CORS header in the policy to the response. CloudFront then sends that response to the viewer. CloudFront ignores the header that it received from the origin.
- When set to `false` and the origin response contains a CORS header (regardless of whether the CORS header is in the policy), CloudFront includes the CORS header that it received from the origin to the response. CloudFront doesn't add any CORS headers in the policy to the response that is sent to the viewer.

Security headers

You can use the security headers settings to add and configure several security-related HTTP response headers in a response headers policy.

This list describes how you can specify settings and valid values in a response headers policy. For more information about each of these headers and how they're used in real-world HTTP responses, see the links to the MDN Web Docs.

Content-Security-Policy

Specifies the content security policy directives that CloudFront uses as values for the `Content-Security-Policy` response header.

For more information about this header and valid policy directives, see Content-Security-Policy in the MDN Web Docs.

> **Note**
>
> The `Content-Security-Policy` header value is limited to 1783 characters.

Referrer-Policy

Specifies the referrer policy directive that CloudFront uses as the value for the `Referrer-Policy` response header. Valid values for this setting are `no-referrer`, `no-referrer-when-downgrade`, `origin`, `origin-when-cross-origin`, `same-origin`, `strict-origin`, `strict-origin-when-cross-origin`, and `unsafe-url`.

For more information about this header and these directives, see Referrer-Policy in the MDN Web Docs.

Strict-Transport-Security

Specifies the directives and settings that CloudFront uses as the value for the `Strict-Transport-Security` response header. For this setting, you separately specify:

- A number of seconds, which CloudFront uses as the value for the `max-age` directive of this header
- A Boolean setting (`true` or `false`) for `preload`, which determines whether CloudFront includes the `preload` directive in the value of this header
- A Boolean setting (`true` or `false`) for `includeSubDomains`, which determines whether CloudFront includes the `includeSubDomains` directive in the value of this header

For more information about this header and these directives, see Strict-Transport-Security in the MDN Web Docs.

X-Content-Type-Options

This is a Boolean setting (`true` or `false`) that determines if CloudFront adds the `X-Content-Type-Options` header to responses. When this setting is `true`, CloudFront adds the `X-Content-Type-Options: nosniff` header to responses. Otherwise CloudFront doesn't add this header.

For more information about this header, see X-Content-Type-Options in the MDN Web Docs.

X-Frame-Options

Specifies the directive that CloudFront uses as the value for the `X-Frame-Options` response header. Valid values for this setting are `DENY` or `SAMEORIGIN`.

For more information about this header and these directives, see X-Frame-Options in the MDN Web Docs.

X-XSS-Protection

Specifies the directives and settings that CloudFront uses as the value for the `X-XSS-Protection` response header. For this setting, you separately specify:

- An `X-XSS-Protection` setting of `0` (disables XSS filtering) or `1` (enables XSS filtering)
- A Boolean setting (`true` or `false`) for `block`, which determines whether CloudFront includes the `mode=block` directive in the value for this header
- A reporting URI, which determines whether CloudFront includes the `report=`*`reporting URI`* directive in the value for this header

You can specify `true` for `block`, or you can specify a reporting URI, but you can't specify both together. For more information about this header and these directives, see X-XSS-Protection in the MDN Web Docs.

Origin override

Each of these security headers settings contains a Boolean setting (`true` or `false`) that determines how CloudFront behaves when the response from the origin contains that header.

When this setting is set to `true` and the origin response contains the header, CloudFront adds the header in the policy to the response that it sends to the viewer. It ignores the header that it received from the origin.

When this setting is set to `false` and the origin response contains the header, CloudFront includes the header that it received from the origin in the response that it sends to the viewer.

When the origin response doesn't contain the header, CloudFront adds the header in the policy to the response that it sends to the viewer. CloudFront does this when this setting is set to `true` or `false`.

Custom headers

You can use custom headers settings to add and configure custom HTTP headers in a response headers policy. CloudFront adds these headers to every response that it returns to viewers. For each custom header, you also specify the value for the header, though specifying a value is optional. This is because CloudFront can add a response header with no value.

Each custom header also has its own **Origin override** setting:

- When this setting is set to `true` and the origin response contains the custom header that's in the policy, CloudFront adds the custom header in the policy to the response that it sends to the viewer. It ignores the header that it received from the origin.
- When this setting is `false` and the origin response contains the custom header that's in the policy, CloudFront includes the custom header that it received from the origin in the response that it sends to the viewer.
- When the origin response doesn't contain the custom header that's in the policy, CloudFront adds the custom header in the policy to the response that it sends to the viewer. CloudFront does this when this setting is set to `true` or `false`.

Remove headers

You can specify headers that you want CloudFront to remove from the responses it receives from the origin so the headers are not included in the responses that CloudFront sends to viewers. CloudFront removes the headers from every response it sends to viewers, whether the objects is served from CloudFront cache or from the origin. For example, you can remove headers that are of no use to browsers, such as `X-Powered-By` or `Vary`, so that CloudFront removes these headers from the responses that it sends to viewers.

When you specify headers to remove using a response headers policy, CloudFront removes the headers first and then adds any headers that are specified in other sections of the response headers policy (CORS headers, security headers, custom headers, and so on). If you specify a header to remove but also add the same header in another section of the policy, CloudFront includes the header in the responses that it sends to viewers.

> **Note**
>
> You can use a response headers policy to remove the `Server` and `Date` headers that CloudFront received from the origin, so that these headers (as received from the origin) are

not included in the responses that CloudFront sends to viewers. However, if you do that, CloudFront adds its own version of these headers to responses that it sends to viewers. For the `Server` header that CloudFront adds, the header's value is `CloudFront`.

Headers that you can't remove

You cannot remove the following headers using a response headers policy. If you specify these headers in the **Remove headers** section of a response headers policy (`ResponseHeadersPolicyRemoveHeadersConfig` in the API), you receive an error.

- `Connection`
- `Content-Encoding`
- `Content-Length`
- `Expect`
- `Host`
- `Keep-Alive`
- `Proxy-Authenticate`
- `Proxy-Authorization`
- `Proxy-Connection`
- `Trailer`
- `Transfer-Encoding`
- `Upgrade`
- `Via`
- `Warning`
- `X-Accel-Buffering`
- `X-Accel-Charset`
- `X-Accel-Limit-Rate`
- `X-Accel-Redirect`
- `X-Amz-Cf-.*`
- `X-Amzn-Auth`
- `X-Amzn-Cf-Billing`
- `X-Amzn-Cf-Id`

- X-Amzn-Cf-Xff
- X-Amzn-ErrorType
- X-Amzn-Fle-Profile
- X-Amzn-Header-Count
- X-Amzn-Header-Order
- X-Amzn-Lambda-Integration-Tag
- X-Amzn-RequestId
- X-Edge-.*
- X-Forwarded-Proto
- X-Real-Ip

Server-Timing header

Use the Server-Timing header setting to enable the Server-Timing header in HTTP responses sent from CloudFront. You can use this header to view metrics that can help you gain insights about the behavior and performance of CloudFront and your origin. For example, you can see which cache layer served a cache hit. Or, you can see the first byte latency from the origin if there's a cache miss. The metrics in the Server-Timing header can help you troubleshoot issues or test the efficiency of your CloudFront or origin configuration.

For more information about using the Server-Timing header with CloudFront, see the following topics.

To enable the Server-Timing header, create (or edit) a response headers policy.

Topics

- Sampling rate and Pragma request header
- Server-Timing header from the origin
- Server-Timing header metrics
- Server-Timing header examples

Sampling rate and Pragma request header

When you enable the Server-Timing header in a response headers policy, you also specify the *sampling rate*. The sampling rate is a number 0–100 (inclusive) that specifies the percentage of

responses that you want CloudFront to add the `Server-Timing` header to. When you set the sampling rate to 100, CloudFront adds the `Server-Timing` header to the HTTP response for every request that matches the cache behavior that the response headers policy is attached to. When you set it to 50, CloudFront adds the header to 50% of the responses for requests that match the cache behavior. You can set the sampling rate to any number 0–100 with up to four decimal places.

When the sampling rate is set to a number lower than 100, you can't control which responses CloudFront adds the `Server-Timing` header to, only the percentage. However, you can add the `Pragma` header with a value set to `server-timing` in an HTTP request to receive the `Server-Timing` header in the response to that request. This works no matter what the sampling rate is set to. Even when the sampling rate is set to zero (0), CloudFront adds the `Server-Timing` header to the response if the request contains the `Pragma: server-timing` header.

Server-Timing header from the origin

When there is a cache miss and CloudFront forwards the request to the origin, the origin might include a `Server-Timing` header in its response to CloudFront. In this case, CloudFront adds its metrics to the `Server-Timing` header that it received from the origin. The response that CloudFront sends to the viewer contains a single `Server-Timing` header that includes the value that came from the origin and the metrics that CloudFront added. The header value from the origin might be at the end, or in between two sets of metrics that CloudFront adds to the header.

When there is a cache hit, the response that CloudFront sends to the viewer contains a single `Server-Timing` header that includes only the CloudFront metrics in the header value (the value from the origin is not included).

Server-Timing header metrics

When CloudFront adds the `Server-Timing` header to an HTTP response, the value of the header contains one or more metrics that can help you gain insights about the behavior and performance of CloudFront and your origin. The following list contains all the metrics and their potential values. A `Server-Timing` header contains only some of these metrics, depending on the nature of the request and response through CloudFront.

Some of these metrics are included in the `Server-Timing` header with a name only (no value). Others are a name and a value. When a metric has a value, the name and value are separated by a semicolon (`;`). When the header contains more than one metric, the metrics are separated by a comma (`,`).

cdn-cache-hit

CloudFront provided a response from the cache without making a request to the origin.

cdn-cache-refresh

CloudFront provided a response from the cache after sending a request to the origin to verify that the cached object is still valid. In this case, CloudFront didn't retrieve the full object from the origin.

cdn-cache-miss

CloudFront didn't provide the response from the cache. In this case, CloudFront requested the full object from the origin before returning the response.

cdn-pop

Contains a value that describes which CloudFront point of presence (POP) handled the request.

cdn-rid

Contains a value with the CloudFront unique identifier for the request. You can use this request identifier (RID) when troubleshooting issues with Support

cdn-hit-layer

This metric is present when CloudFront provides a response from the cache without making a request to the origin. It contains one of the following values:

- **EDGE** – CloudFront provided the cached response from a POP location.
- **REC** – CloudFront provided the cached response from a regional edge cache (REC) location.
- **Origin Shield** – CloudFront provided the cached response from the REC that's acting as Origin Shield.

cdn-upstream-layer

When CloudFront requests the full object from the origin, this metric is present and contains one of the following values:

- **EDGE** – A POP location sent the request directly to the origin.
- **REC** – A REC location sent the request directly to the origin.
- **Origin Shield** – The REC that's acting as Origin Shield sent the request directly to the origin.

cdn-upstream-dns

Contains a value with the number of milliseconds that were spent retrieving the DNS record for the origin. A value of zero (0) indicates that CloudFront used a cached DNS result or reused an existing connection.

cdn-upstream-connect

Contains a value with the number of milliseconds between when the origin DNS request completed and a TCP (and TLS, if applicable) connection to the origin completed. A value of zero (0) indicates that CloudFront reused an existing connection.

cdn-upstream-fbl

Contains a value with the number of milliseconds between when the origin HTTP request is completed and when the first byte is received in the response from the origin (first byte latency).

cdn-downstream-fbl

Contains a value with the number of milliseconds between when the edge location finished receiving the request and when it sent the first byte of the response to the viewer.

Server-Timing header examples

The following are examples of a `Server-Timing` header that a viewer might receive from CloudFront when the `Server-Timing` header setting is enabled.

Example– cache miss

The following example shows a `Server-Timing` header that a viewer might receive when the requested object is not in the CloudFront cache.

```
Server-Timing: cdn-upstream-layer;desc="EDGE",cdn-upstream-dns;dur=0,cdn-upstream-
connect;dur=114,cdn-upstream-fbl;dur=177,cdn-cache-miss,cdn-pop;desc="PHX50-C2",cdn-
rid;desc="yNPsyYn7skvTzwWkq3Wcc8Nj_foxUjQUe9H1ifslzWhb0w7aLbFvGg==",cdn-downstream-
fbl;dur=436
```

This `Server-Timing` header indicates the following:

- The origin request was sent from a CloudFront point of presence (POP) location (`cdn-upstream-layer;desc="EDGE"`).

- CloudFront used a cached DNS result for the origin (`cdn-upstream-dns;dur=0`).
- It took 114 milliseconds for CloudFront to complete the TCP (and TLS, if applicable) connection to the origin (`cdn-upstream-connect;dur=114`).
- It took 177 milliseconds for CloudFront to receive the first byte of the response from the origin, after completing the request (`cdn-upstream-fbl;dur=177`).
- The requested object wasn't in CloudFront cache (`cdn-cache-miss`).
- The request was received at the edge location identified by the code `PHX50-C2` (`cdn-pop;desc="PHX50-C2"`).
- The CloudFront unique ID for this request was `yNPsyYn7skvTzwWkq3Wcc8Nj_foxUjQUe9H1ifslzWhb0w7aLbFvGg==` (`cdn-rid;desc="yNPsyYn7skvTzwWkq3Wcc8Nj_foxUjQUe9H1ifslzWhb0w7aLbFvGg=="`).
- It took 436 milliseconds for CloudFront to send the first byte of the response to the viewer, after receiving the viewer request (`cdn-downstream-fbl;dur=436`).

Example– cache hit

The following example shows a `Server-Timing` header that a viewer might receive when the requested object is in CloudFront cache.

```
Server-Timing: cdn-cache-hit,cdn-pop;desc="SEA19-C1",cdn-
rid;desc="nQBz4aJU2kP9iC3KHEq7vFxfMozu-VYBwGzkW9diOpeVc7xsrLKj-g==",cdn-hit-
layer;desc="REC",cdn-downstream-fbl;dur=137
```

This `Server-Timing` header indicates the following:

- The requested object was in the cache (`cdn-cache-hit`).
- The request was received at the edge location identified by the code `SEA19-C1` (`cdn-pop;desc="SEA19-C1"`).
- The CloudFront unique ID for this request was `nQBz4aJU2kP9iC3KHEq7vFxfMozu-VYBwGzkW9diOpeVc7xsrLKj-g==` (`cdn-rid;desc="nQBz4aJU2kP9iC3KHEq7vFxfMozu-VYBwGzkW9diOpeVc7xsrLKj-g=="`).
- The requested object was cached in a regional edge cache (REC) location (`cdn-hit-layer;desc="REC"`).
- It took 137 milliseconds for CloudFront to send the first byte of the response to the viewer, after receiving the viewer request (`cdn-downstream-fbl;dur=137`).

Create response headers policies

You can use a response headers policy to specify the HTTP headers that Amazon CloudFront adds or removes in HTTP responses. For more information about response headers policies and reasons to use them, see *Add or remove response headers with a policy*.

You can create a response headers policy in the CloudFront console. Or you can create one by using AWS CloudFormation, the AWS Command Line Interface (AWS CLI), or the CloudFront API. After you create a response headers policy, you attach it to one or more cache behaviors in a CloudFront distribution.

Before you create a custom response headers policy, check if one of the managed response headers policies fits your use case. If one does, you can attach it to your cache behavior. That way, you don't need to create or manage your own response headers policy.

Console

To create a response headers policy (console)

1. Sign in to the AWS Management Console, then go to the **Response headers** tab on the **Policies** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/policies/responseHeaders.
2. Choose **Create response headers policy**.
3. In the **Create response headers policy** form, do the following:

   a. In the **Details** panel, enter a **Name** for the response headers policy and (optionally) a **Description** that explains what the policy is for.

   b. In the **Cross-origin resource sharing (CORS)** panel, choose the **Configure CORS** toggle and configure any CORS headers that you want to add to the policy. If you want the configured headers to override the headers that CloudFront receives from the origin, select the **Origin override** check box.

      For more information about the CORS headers settings, see the section called “CORS headers”.

   c. In the **Security headers** panel, choose the toggle and configure each of the security headers that you want to add to the policy.

      For more information about the security headers settings, see the section called “Security headers”.

d. In the **Custom headers** panel, add any custom headers that you want to include in the policy.

   For more information about the custom headers settings, see the section called "Custom headers".

e. In the **Remove headers** panel, add the names of any headers that you want CloudFront to remove from the origin's response and not include in the response that CloudFront sends to viewers.

   For more information about the remove headers settings, see the section called "Remove headers".

f. In the **Server-Timing header** panel, choose the **Enable** toggle and enter a sampling rate (a number between 0 and 100, inclusive).

   For more information about the `Server-Timing` header, see the section called "Server-Timing header".

4. Choose **Create** to create the policy.

After you create a response headers policy, you can attach it to a cache behavior in a CloudFront distribution.

To attach a response headers policy to an existing distribution (console)

1. Open the **Distributions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/distributions.
2. Choose the distribution to update, then choose the **Behaviors** tab.
3. Select the cache behavior to update, then choose **Edit**.

   Or, to create a new cache behavior, choose **Create behavior**.
4. For **Response headers policy**, choose the policy to add to the cache behavior.
5. Choose **Save changes** to update the cache behavior. If you're creating a new cache behavior, choose **Create behavior**.

To attach a response headers policy to a new distribution (console)

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. Choose **Create distribution**.
3. For **Response headers policy**, choose the policy to add to the cache behavior.
4. Choose the other settings for your distribution. For more information, see the section called "All distribution settings".
5. Choose **Create distribution** to create the distribution.

CloudFormation

To create a response headers policy with CloudFormation, use the `AWS::CloudFront::ResponseHeadersPolicy` resource type. The following example shows the CloudFormation template syntax, in YAML format, for creating a response headers policy.

```
Type: AWS::CloudFront::ResponseHeadersPolicy
Properties:
  ResponseHeadersPolicyConfig:
    Name: EXAMPLE-Response-Headers-Policy
    Comment: Example response headers policy for the documentation
    CorsConfig:
      AccessControlAllowCredentials: false
      AccessControlAllowHeaders:
        Items:
          - '*'
      AccessControlAllowMethods:
        Items:
          - GET
          - OPTIONS
      AccessControlAllowOrigins:
        Items:
          - https://example.com
          - https://docs.example.com
      AccessControlExposeHeaders:
        Items:
          - '*'
      AccessControlMaxAgeSec: 600
      OriginOverride: false
    CustomHeadersConfig:
      Items:
        - Header: Example-Custom-Header-1
          Value: value-1
          Override: true
        - Header: Example-Custom-Header-2
```

```
          Value: value-2
          Override: true
    SecurityHeadersConfig:
      ContentSecurityPolicy:
        ContentSecurityPolicy: default-src 'none'; img-src 'self'; script-src
 'self'; style-src 'self'; object-src 'none'; frame-ancestors 'none'
        Override: false
      ContentTypeOptions: # You don't need to specify a value for 'X-Content-Type-
Options'.
                          # Simply including it in the template sets its value to
 'nosniff'.
        Override: false
      FrameOptions:
        FrameOption: DENY
        Override: false
      ReferrerPolicy:
        ReferrerPolicy: same-origin
        Override: false
      StrictTransportSecurity:
        AccessControlMaxAgeSec: 63072000
        IncludeSubdomains: true
        Preload: true
        Override: false
      XSSProtection:
        ModeBlock: true # You can set ModeBlock to 'true' OR set a value for
 ReportUri, but not both
        Protection: true
        Override: false
    ServerTimingHeadersConfig:
      Enabled: true
      SamplingRate: 50
    RemoveHeadersConfig:
      Items:
        - Header: Vary
        - Header: X-Powered-By
```

For more information, see AWS::CloudFront::ResponseHeadersPolicy in the *AWS CloudFormation User Guide*.

CLI

To create a response headers policy with the AWS Command Line Interface (AWS CLI), use the **aws cloudfront create-response-headers-policy** command. You can use an input file to provide

the input parameters for the command, rather than specifying each individual parameter as command line input.

To create a response headers policy (CLI with input file)

1. Use the following command to create a file that's named `response-headers-policy.yaml`. This file contains all of the input parameters for the **create-response-headers-policy** command.

   ```
   aws cloudfront create-response-headers-policy --generate-cli-skeleton yaml-input
    > response-headers-policy.yaml
   ```

2. Open the `response-headers-policy.yaml` file that you just created. Edit the file to specify a policy name and the desired response headers policy configuration, then save the file.

   For more information about the response headers policy settings, see the section called "Understand response headers policies".

3. Use the following command to create the response headers policy. The policy that you create uses the input parameters from the `response-headers-policy.yaml` file.

   ```
   aws cloudfront create-response-headers-policy --cli-input-yaml file://response-
   headers-policy.yaml
   ```

   Make note of the `Id` value in the command output. This is the response headers policy ID. You need it to attach the policy to the cache behavior of a CloudFront distribution.

To attach a response headers policy to an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to update. Replace *`distribution_ID`* with the distribution ID.

   ```
   aws cloudfront get-distribution-config --id distribution_ID --output yaml >
    dist-config.yaml
   ```

2. Open the file that's named `dist-config.yaml` that you just created. Edit the file, making the following changes to the cache behavior to make it use the response headers policy.

   - In the cache behavior, add a field that's named `ResponseHeadersPolicyId`. For the field's value, use the response headers policy ID that you noted after creating the policy.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the distribution to use the response headers policy. Replace *`distribution_ID`* with the distribution ID.

```
aws cloudfront update-distribution --id distribution_ID --cli-input-yaml file://
dist-config.yaml
```

To attach a response headers policy to a new distribution (CLI with input file)

1. Use the following command to create a file that's named `distribution.yaml`. This file contains all of the input parameters for the **create-distribution** command.

```
aws cloudfront create-distribution --generate-cli-skeleton yaml-input >
 distribution.yaml
```

2. Open the `distribution.yaml` file that you just created. In the default cache behavior, in the `ResponseHeadersPolicyId` field, enter the response headers policy ID that you noted after creating the policy. Continue editing the file to specify the distribution settings that you want, then save the file when finished.

   For more information about the distribution settings, see All distribution settings reference.

3. Use the following command to create the distribution using input parameters from the `distribution.yaml` file.

```
aws cloudfront create-distribution --cli-input-yaml file://distribution.yaml
```

API

To create a response headers policy with the CloudFront API, use CreateResponseHeadersPolicy. For more information about the fields that you specify in this API call, see the section called "Understand response headers policies" and the API reference documentation for your AWS SDK or other API client.

After you create a response headers policy, you can attach it to a cache behavior, using one of the following API calls:

- To attach it to a cache behavior in an existing distribution, use UpdateDistribution.
- To attach it to a cache behavior in a new distribution, use CreateDistribution.

For both of these API calls, provide the response headers policy ID in the `ResponseHeadersPolicyId` field, inside a cache behavior. For more information about the other fields that you specify in these API calls, see All distribution settings reference and the API reference documentation for your AWS SDK or other API client.

Use managed response headers policies

With a CloudFront response headers policy, you can specify the HTTP headers that Amazon CloudFront removes or adds in responses that it sends to viewers. For more information about response headers policies and reasons to use them, see *Add or remove response headers with a policy*.

CloudFront provides managed response headers policies that you can attach to cache behaviors in your CloudFront distributions. With a managed response headers policy, you don't need to write or maintain your own policy. The managed policies contain sets of HTTP response headers for common use cases.

To use a managed response headers policy, you attach it to a cache behavior in your distribution. The process is the same as when you create a custom response headers policy. However, instead of creating a new policy, you attach one of the managed policies. You attach the policy either by name (with the console) or by ID (with CloudFormation, the AWS CLI, or the AWS SDKs). The names and IDs are listed in the following section.

For more information, see the section called "Create response headers policies".

The following topics describe the managed response headers policies that you can use.

Topics

- CORS-and-SecurityHeadersPolicy
- CORS-With-Preflight
- CORS-with-preflight-and-SecurityHeadersPolicy
- SecurityHeadersPolicy
- SimpleCORS

CORS-and-SecurityHeadersPolicy

View this policy in the CloudFront console

Use this managed policy to allow simple CORS requests from any origin. This policy also adds a set of security headers to all responses that CloudFront sends to viewers. This policy combines the the section called “SimpleCORS” and the section called “SecurityHeadersPolicy” policies into one.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`e61eb60c-9c35-4d20-a928-2b84e02af89c`

Policy settings

	Header name	Header value	Override origin?
CORS headers:	`Access-Control-Allow-Origin`	`*`	No
Security headers:	`Referrer-Policy`	`strict-origin-when-cross-origin`	No
	`Strict-Transport-Security`	`max-age=31536000`	No
	`X-Content-Type-Options`	`nosniff`	Yes
	`X-Frame-Options`	`SAMEORIGIN`	No
	`X-XSS-Protection`	`1; mode=block`	No

CORS-With-Preflight

View this policy in the CloudFront console

Use this managed policy to allow CORS requests from any origin, including preflight requests. For preflight requests (using the HTTP `OPTIONS` method), CloudFront adds all three of the following headers to the response. For simple CORS requests, CloudFront adds only the `Access-Control-Allow-Origin` header.

If the response that CloudFront receives from the origin includes any of these headers, CloudFront uses the received header (and its value) in its response to the viewer. CloudFront doesn't use the header in this policy.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`5cc3b908-e619-4b99-88e5-2cf7f45965bd`

Policy settings

	Header name	Header value	Override origin?
CORS headers:	`Access-Control-Allow-Methods`	`DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT`	No
	`Access-Control-Allow-Origin`	`*`	
	`Access-Control-Expose-Headers`	`*`	

CORS-with-preflight-and-SecurityHeadersPolicy

View this policy in the CloudFront console

Use this managed policy to allow CORS requests from any origin. This includes preflight requests. This policy also adds a set of security headers to all responses that CloudFront sends to viewers. This policy combines the the section called “CORS-With-Preflight” and the section called “SecurityHeadersPolicy” policies into one.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`eaab4381-ed33-4a86-88ca-d9558dc6cd63`

Policy settings

	Header name	Header value	Override origin?
CORS headers:	`Access-Control-Allow-Methods`	`DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT`	No
	`Access-Control-Allow-Origin`	`*`	
	`Access-Control-Expose-Headers`	`*`	
Security headers:	`Referrer-Policy`	`strict-origin-when-cross-origin`	No
	`Strict-Transport-Security`	`max-age=31536000`	No
	`X-Content-Type-Options`	`nosniff`	Yes
	`X-Frame-Options`	`SAMEORIGIN`	No
	`X-XSS-Protection`	`1; mode=block`	No

SecurityHeadersPolicy

View this policy in the CloudFront console

Use this managed policy to add a set of security headers to all responses that CloudFront sends to viewers. For more information about these security headers, see Mozilla's web security guidelines.

With this response headers policy, CloudFront adds `X-Content-Type-Options: nosniff` to all responses. This is the case when the response that CloudFront received from the origin included this header and when it didn't. For all other headers in this policy, if the response that CloudFront receives from the origin includes the header, CloudFront uses the received header (and its value) in its response to the viewer. It doesn't use the header in this policy.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`67f7725c-6f97-4210-82d7-5512b31e9d03`

Policy settings

	Header name	Header value	Override origin?
Security headers:	`Referrer-Policy`	`strict-origin-when-cross-origin`	No
	`Strict-Transport-Security`	`max-age=31536000`	No
	`X-Content-Type-Options`	`nosniff`	Yes
	`X-Frame-Options`	`SAMEORIGIN`	No
	`X-XSS-Protection`	`1; mode=block`	No

SimpleCORS

View this policy in the CloudFront console

Use this managed policy to allow simple CORS requests from any origin. With this policy, CloudFront adds the header `Access-Control-Allow-Origin: *` to all responses for simple CORS requests.

If the response that CloudFront receives from the origin includes the `Access-Control-Allow-Origin` header, CloudFront uses that header (and its value) in its response to the viewer. CloudFront doesn't use the header in this policy.

When using CloudFormation, the AWS CLI, or the CloudFront API, the ID for this policy is:

`60669652-455b-4ae9-85a4-c4c02393f86c`

Policy settings

	Header name	Header value	Override origin?
CORS headers:	`Access-Control-Allow-Origin`	*	No

Request and response behavior

The following topics describe how CloudFront handles requests and responses.

You can learn about how CloudFront interacts with Amazon S3 or custom origins, handles various HTTP methods and headers, processes status codes, and manages caching and error responses.

Topics

- How CloudFront processes HTTP and HTTPS requests
- Request and response behavior for Amazon S3 origins
- Request and response behavior for custom origins
- Request and response behavior for origin groups
- Add custom headers to origin requests
- How CloudFront processes partial requests for an object (range GETs)
- How CloudFront processes HTTP 3xx status codes from your origin
- How CloudFront processes HTTP 4xx and 5xx status codes from your origin
- Generate custom error responses

How CloudFront processes HTTP and HTTPS requests

For Amazon S3 origins, CloudFront accepts requests in both HTTP and HTTPS protocols for objects in a CloudFront distribution by default. CloudFront then forwards the requests to your Amazon S3 bucket using the same protocol in which the requests were made.

For custom origins, when you create your distribution, you can specify how CloudFront accesses your origin: HTTP only, or matching the protocol that is used by the viewer. For more information about how CloudFront handles HTTP and HTTPS requests for custom origins, see Protocols.

For information about how to restrict your distribution so that end users can only access objects using HTTPS, see Use HTTPS with CloudFront.

> **Note**
>
> The charge for HTTPS requests is higher than the charge for HTTP requests. For more information about billing rates, see CloudFront pricing.

Request and response behavior for Amazon S3 origins

To understand how CloudFront processes requests and responses when you're using Amazon S3 as your origin, see the following sections:

Topics

- How CloudFront processes and forwards requests to your Amazon S3 origin
- How CloudFront processes responses from your Amazon S3 origin

How CloudFront processes and forwards requests to your Amazon S3 origin

Learn about how CloudFront processes viewer requests and forwards the requests to your Amazon S3 origin.

Contents

- Caching duration and minimum TTL
- Client IP addresses
- Conditional GET requests
- Cookies
- Cross-origin resource sharing (CORS)
- GET requests that include a body
- HTTP methods
- HTTP request headers that CloudFront removes or updates
- Maximum length of a request and maximum length of a URL
- OCSP stapling
- Protocols
- Query strings
- Origin connection timeout and attempts
- Origin response timeout
- Simultaneous requests for the same object (request collapsing)

Caching duration and minimum TTL

To control how long your objects stay in a CloudFront cache before CloudFront forwards another request to your origin, you can:

- Configure your origin to add a `Cache-Control` or an `Expires` header field to each object.
- Specify a value for Minimum TTL in CloudFront cache behaviors.
- Use the default value of 24 hours.

For more information, see Manage how long content stays in the cache (expiration).

Client IP addresses

If a viewer sends a request to CloudFront and doesn't include an `X-Forwarded-For` request header, CloudFront gets the IP address of the viewer from the TCP connection, adds an `X-Forwarded-For` header that includes the IP address, and forwards the request to the origin. For example, if CloudFront gets the IP address `192.0.2.2` from the TCP connection, it forwards the following header to the origin:

```
X-Forwarded-For: 192.0.2.2
```

If a viewer sends a request to CloudFront and includes an `X-Forwarded-For` request header, CloudFront gets the IP address of the viewer from the TCP connection, appends it to the end of the `X-Forwarded-For` header, and forwards the request to the origin. For example, if the viewer request includes `X-Forwarded-For: 192.0.2.4,192.0.2.3` and CloudFront gets the IP address `192.0.2.2` from the TCP connection, it forwards the following header to the origin:

```
X-Forwarded-For: 192.0.2.4,192.0.2.3,192.0.2.2
```

> **Note**
>
> The `X-Forwarded-For` header contains IPv4 addresses (such as 192.0.2.44) and IPv6 addresses (such as 2001:0db8:85a3::8a2e:0370:7334).
> When parsing IPv6 addresses in the `X-Forwarded-For` header, use standard IP address parsing libraries that can handle any valid RFC 4291 IPv6 format.

Conditional GET requests

When CloudFront receives a request for an object that has expired from an edge cache, it forwards the request to the Amazon S3 origin to get the latest version of the object or to get confirmation from Amazon S3 that the CloudFront edge cache already has the latest version. When Amazon S3 originally sent the object to CloudFront, it included an `ETag` value and a `LastModified` value in the response. In the new request that CloudFront forwards to Amazon S3, CloudFront adds one or both of the following headers:

- An `If-Match` or `If-None-Match` header that contains the `ETag` value for the expired version of the object.
- An `If-Modified-Since` header that contains the `LastModified` value for the expired version of the object.

Amazon S3 uses this information to determine whether the object has been updated and, therefore, whether to return the entire object to CloudFront or to return only an HTTP 304 status code (not modified).

Cookies

Amazon S3 doesn't process cookies. If you configure a cache behavior to forward cookies to an Amazon S3 origin, CloudFront forwards the cookies, but Amazon S3 ignores them. All future requests for the same object, regardless if you vary the cookie, are served from the existing object in the cache.

Cross-origin resource sharing (CORS)

If you want CloudFront to respect Amazon S3 cross-origin resource sharing settings, configure CloudFront to forward selected headers to Amazon S3. For more information, see Cache content based on request headers.

GET requests that include a body

If a viewer `GET` request includes a body, CloudFront returns an HTTP status code 403 (Forbidden) to the viewer.

HTTP methods

If you configure CloudFront to process all of the HTTP methods that it supports, CloudFront accepts the following requests from viewers and forwards them to your Amazon S3 origin:

- DELETE
- GET
- HEAD
- OPTIONS
- PATCH
- POST
- PUT

CloudFront always caches responses to GET and HEAD requests. You can also configure CloudFront to cache responses to OPTIONS requests. CloudFront doesn't cache responses to requests that use the other methods.

If you want to use multi-part uploads to add objects to an Amazon S3 bucket, you must add a CloudFront origin access control (OAC) to your distribution and give the OAC the needed permissions. For more information, see the section called “Restrict access to an Amazon S3 origin”.

Important

If you configure CloudFront to accept and forward to Amazon S3 all of the HTTP methods that CloudFront supports, you must create a CloudFront OAC to restrict access to your Amazon S3 content and give the OAC the required permissions. For example, if you configure CloudFront to accept and forward these methods because you want to use the PUT method, you must configure Amazon S3 bucket policies to handle DELETE requests appropriately so viewers can't delete resources that you don't want them to. For more information, see the section called “Restrict access to an Amazon S3 origin”.

For information about the operations supported by Amazon S3, see the Amazon S3 documentation.

HTTP request headers that CloudFront removes or updates

CloudFront removes or updates some headers before forwarding requests to your Amazon S3 origin. For most headers this behavior is the same as for custom origins. For a full list of HTTP request headers and how CloudFront processes them, see HTTP request headers and CloudFront behavior (custom and Amazon S3 origins).

Maximum length of a request and maximum length of a URL

The maximum length of a request, including the path, the query string (if any), and headers, is 32,768 bytes.

CloudFront constructs a URL from the request. The maximum length of this URL is 8192 bytes.

If a URL exceeds the maximum length, CloudFront returns HTTP status code 414 (URI Too Long) to the viewer. If a request exceeds the maximum length because the header size is exceeded, CloudFront returns HTTP status code 494 to the viewer. In both cases, CloudFront then terminates the TCP connection to the viewer.

OCSP stapling

When a viewer submits an HTTPS request for an object, CloudFront or the viewer must confirm with the certificate authority (CA) that the SSL certificate for the domain hasn't been revoked. OCSP stapling speeds up certificate validation by allowing CloudFront to validate the certificate and to cache the response from the CA, so the client doesn't need to validate the certificate directly with the CA.

The performance improvement of OCSP stapling is more pronounced when CloudFront receives many HTTPS requests for objects in the same domain. Each server in a CloudFront edge location must submit a separate validation request. When CloudFront receives a lot of HTTPS requests for the same domain, every server in the edge location soon has a response from the CA that it can staple to a packet in the SSL handshake. When the viewer is satisfied that the certificate is valid, CloudFront can serve the requested object. If your distribution doesn't get much traffic in a CloudFront edge location, new requests are more likely to be directed to a server that hasn't validated the certificate with the CA yet. In that case, the viewer separately performs the validation step and the CloudFront server serves the object. That CloudFront server also submits a validation request to the CA, so the next time it receives a request that includes the same domain name, it has a validation response from the CA.

Protocols

CloudFront forwards HTTP or HTTPS requests to the origin server based on the protocol of the viewer request, either HTTP or HTTPS.

> **Important**
>
> If your Amazon S3 bucket is configured as a website endpoint, you can't configure CloudFront to use HTTPS to communicate with your origin because Amazon S3 doesn't support HTTPS connections in that configuration.

Query strings

You can configure whether CloudFront forwards query string parameters to your Amazon S3 origin. For more information, see Cache content based on query string parameters.

Origin connection timeout and attempts

Origin connection timeout is the number of seconds that CloudFront waits when trying to establish a connection to the origin.

Origin connection attempts is the number of times that CloudFront attempts to connect to the origin.

Together, these settings determine how long CloudFront tries to connect to the origin before failing over to the secondary origin (in the case of an origin group) or returning an error response to the viewer. By default, CloudFront waits as long as 30 seconds (3 attempts of 10 seconds each) before attempting to connect to the secondary origin or returning an error response. You can reduce this time by specifying a shorter connection timeout, fewer attempts, or both.

For more information, see Control origin timeouts and attempts.

Origin response timeout

The *origin response timeout*, also known as the *origin read timeout* or *origin request timeout*, applies to both of the following:

- The amount of time, in seconds, that CloudFront waits for a response after forwarding a request to the origin.
- The amount of time, in seconds, that CloudFront waits after receiving a packet of a response from the origin and before receiving the next packet.

CloudFront behavior depends on the HTTP method of the viewer request:

- `GET` and `HEAD` requests – If the origin doesn't respond within 30 seconds or stops responding for 30 seconds, CloudFront drops the connection. If the specified number of origin connection attempts is more than 1, CloudFront tries again to get a complete response. CloudFront tries up to 3 times, as determined by the value of the *origin connection attempts* setting. If the origin doesn't respond during the final attempt, CloudFront doesn't try again until it receives another request for content on the same origin.
- `DELETE`, `OPTIONS`, `PATCH`, `PUT`, and `POST` requests – If the origin doesn't respond within 30 seconds, CloudFront drops the connection and doesn't try again to contact the origin. The client can resubmit the request if necessary.

You can't change the response timeout for an Amazon S3 origin (an S3 bucket that is *not* configured with static website hosting).

Simultaneous requests for the same object (request collapsing)

When a CloudFront edge location receives a request for an object and the object isn't in the cache or the cached object is expired, CloudFront immediately sends the request to the origin. However, if there are simultaneous requests for the same object—that is, if additional requests for the same object (with the same cache key) arrive at the edge location before CloudFront receives the response to the first request—CloudFront pauses before forwarding the additional requests to the origin. This brief pause helps to reduce the load on the origin. CloudFront sends the response from the original request to all the requests that it received while it was paused. This is called *request collapsing*. In CloudFront logs, the first request is identified as a `Miss` in the `x-edge-result-type` field, and the collapsed requests are identified as a `Hit`. For more information about CloudFront logs, see the section called "CloudFront and edge function logging".

CloudFront only collapses requests that share a *cache key*. If the additional requests do not share the same cache key because, for example, you configured CloudFront to cache based on request headers or cookies or query strings, CloudFront forwards all the requests with a unique cache key to your origin.

If you want to prevent all request collapsing, you can use the managed cache policy `CachingDisabled`, which also prevents caching. For more information, see Use managed cache policies.

If you want to prevent request collapsing for specific objects, you can set the minimum TTL for the cache behavior to 0 *and* configure the origin to send `Cache-Control: private`, `Cache-Control: no-store`, `Cache-Control: no-cache`, `Cache-Control: max-age=0`, or `Cache-`

`Control: s-maxage=0`. These configurations will increase the load on your origin and introduce additional latency for the simultaneous requests that are paused while CloudFront waits for the response to the first request.

How CloudFront processes responses from your Amazon S3 origin

Learn about how CloudFront processes responses from your Amazon S3 origin.

Contents

- Canceled requests
- HTTP response headers that CloudFront removes or updates
- Maximum cacheable file size
- Redirects

Canceled requests

If an object is not in the edge cache, and if a viewer terminates a session (for example, closes a browser) after CloudFront gets the object from your origin but before it can deliver the requested object, CloudFront does not cache the object in the edge location.

HTTP response headers that CloudFront removes or updates

CloudFront removes or updates the following header fields before forwarding the response from your Amazon S3 origin to the viewer:

- `X-Amz-Id-2`
- `X-Amz-Request-Id`
- `Set-Cookie` – If you configure CloudFront to forward cookies, it will forward the `Set-Cookie` header field to clients. For more information, see Cache content based on cookies.
- `Trailer`
- `Transfer-Encoding` – If your Amazon S3 origin returns this header field, CloudFront sets the value to `chunked` before returning the response to the viewer.
- `Upgrade`
- `Via` – CloudFront sets the value to the following in the response to the viewer:

```
Via: http-version alphanumeric-string.cloudfront.net (CloudFront)
```

For example, the value is something like the following:

```
Via: 1.1 1026589cc7887e7a0dc7827b4example.cloudfront.net (CloudFront)
```

Maximum cacheable file size

The maximum size of a response body that CloudFront saves in its cache is 50 GB. This includes chunked transfer responses that don't specify the `Content-Length` header value.

You can use CloudFront to cache an object that is larger than this size by using range requests to request the objects in parts that are each 50 GB or smaller. CloudFront caches these parts because each of them is 50 GB or smaller. After the viewer retrieves all the parts of the object, it can reconstruct the original, larger object. For more information, see Use range requests to cache large objects.

Redirects

You can configure an Amazon S3 bucket to redirect all requests to another host name; this can be another Amazon S3 bucket or an HTTP server. If you configure a bucket to redirect all requests and if the bucket is the origin for a CloudFront distribution, we recommend that you configure the bucket to redirect all requests to a CloudFront distribution using either the domain name for the distribution (for example, d111111abcdef8.cloudfront.net) or an alternate domain name (a CNAME) that is associated with a distribution (for example, example.com). Otherwise, viewer requests bypass CloudFront, and the objects are served directly from the new origin.

> **Note**
>
> If you redirect requests to an alternate domain name, you must also update the DNS service for your domain by adding a CNAME record. For more information, see Use custom URLs by adding alternate domain names (CNAMEs).

Here's what happens when you configure a bucket to redirect all requests:

1. A viewer (for example, a browser) requests an object from CloudFront.
2. CloudFront forwards the request to the Amazon S3 bucket that is the origin for your distribution.
3. Amazon S3 returns an HTTP status code 301 (Moved Permanently) as well as the new location.

4. CloudFront caches the redirect status code and the new location, and returns the values to the viewer. CloudFront does not follow the redirect to get the object from the new location.
5. The viewer sends another request for the object, but this time the viewer specifies the new location that it got from CloudFront:

   - If the Amazon S3 bucket is redirecting all requests to a CloudFront distribution, using either the domain name for the distribution or an alternate domain name, CloudFront requests the object from the Amazon S3 bucket or the HTTP server in the new location. When the new location returns the object, CloudFront returns it to the viewer and caches it in an edge location.
   - If the Amazon S3 bucket is redirecting requests to another location, the second request bypasses CloudFront. The Amazon S3 bucket or the HTTP server in the new location returns the object directly to the viewer, so the object is never cached in a CloudFront edge cache.

Request and response behavior for custom origins

To understand how CloudFront processes requests and responses when you're using custom origins, see the following sections:

Topics

- How CloudFront processes and forwards requests to your custom origin
- How CloudFront processes responses from your custom origin

How CloudFront processes and forwards requests to your custom origin

Learn about how CloudFront processes viewer requests and forwards the requests to your custom origin.

Contents

- Authentication
- Caching duration and minimum TTL
- Client IP addresses
- Client-side SSL authentication
- Compression
- Conditional requests

- Cookies
- Cross-origin resource sharing (CORS)
- Encryption
- GET requests that include a body
- HTTP methods
- HTTP request headers and CloudFront behavior (custom and Amazon S3 origins)
- HTTP version
- Maximum length of a request and maximum length of a URL
- OCSP stapling
- Persistent connections
- Protocols
- Query strings
- Origin connection timeout and attempts
- Origin response timeout
- Simultaneous requests for the same object (request collapsing)
- User-Agent header

Authentication

If you forward the `Authorization` header to your origin, you can then configure your origin server to request client authentication for the following types of requests:

- `DELETE`
- `GET`
- `HEAD`
- `PATCH`
- `PUT`
- `POST`

For `OPTIONS` requests, client authentication can *only* be configured if you use the following CloudFront settings:

- CloudFront is configured to forward the `Authorization` header to your origin

- CloudFront is configured to *not* cache the response to `OPTIONS` requests

For more information, see Configure CloudFront to forward the `Authorization` header.

You can use HTTP or HTTPS to forward requests to your origin server. For more information, see Use HTTPS with CloudFront.

Caching duration and minimum TTL

To control how long your objects stay in a CloudFront cache before CloudFront forwards another request to your origin, you can:

- Configure your origin to add a `Cache-Control` or an `Expires` header field to each object.
- Specify a value for Minimum TTL in CloudFront cache behaviors.
- Use the default value of 24 hours.

For more information, see Manage how long content stays in the cache (expiration).

Client IP addresses

If a viewer sends a request to CloudFront and does not include an `X-Forwarded-For` request header, CloudFront gets the IP address of the viewer from the TCP connection, adds an `X-Forwarded-For` header that includes the IP address, and forwards the request to the origin. For example, if CloudFront gets the IP address `192.0.2.2` from the TCP connection, it forwards the following header to the origin:

```
X-Forwarded-For: 192.0.2.2
```

If a viewer sends a request to CloudFront and includes an `X-Forwarded-For` request header, CloudFront gets the IP address of the viewer from the TCP connection, appends it to the end of the `X-Forwarded-For` header, and forwards the request to the origin. For example, if the viewer request includes `X-Forwarded-For: 192.0.2.4,192.0.2.3` and CloudFront gets the IP address `192.0.2.2` from the TCP connection, it forwards the following header to the origin:

```
X-Forwarded-For: 192.0.2.4,192.0.2.3,192.0.2.2
```

Some applications, such as load balancers (including Elastic Load Balancing), web application firewalls, reverse proxies, intrusion prevention systems, and API Gateway, append the IP address of the CloudFront edge server that forwarded the request onto the end of the `X-Forwarded-For`

header. For example, if CloudFront includes `X-Forwarded-For: 192.0.2.2` in a request that it forwards to ELB and if the IP address of the CloudFront edge server is 192.0.2.199, the request that your EC2 instance receives contains the following header:

```
X-Forwarded-For: 192.0.2.2,192.0.2.199
```

Note

The `X-Forwarded-For` header contains IPv4 addresses (such as 192.0.2.44) and IPv6 addresses (such as 2001:0db8:85a3::8a2e:0370:7334).
When parsing IPv6 addresses in the `X-Forwarded-For` header, use standard IP address parsing libraries that can handle any valid RFC 4291 IPv6 format.
Also note that the `X-Forwarded-For` header might be modified by every node on the path to the current server (CloudFront). For more information, see section 8.1 in RFC 7239.
You can also modify the header using CloudFront edge compute functions.

Client-side SSL authentication

CloudFront supports mutual TLS (mTLS) authentication where both the client and server authenticate each other using certificates. With mTLS configured, CloudFront can validate client certificates during the TLS handshake and optionally run CloudFront Functions to implement custom validation logic.

For origins that request client-side certificates when mTLS is not configured, CloudFront drops the request.

For more information about configuring mTLS, see ???.

CloudFront does not support client authentication with client-side SSL certificates. If an origin requests a client-side certificate, CloudFront drops the request.

Compression

For more information, see Serve compressed files.

Conditional requests

When CloudFront receives a request for an object that has expired from an edge cache, it forwards the request to the origin either to get the latest version of the object or to get confirmation from the origin that the CloudFront edge cache already has the latest version. Typically, when the origin

last sent the object to CloudFront, it included an `ETag` value, a `LastModified` value, or both values in the response. In the new request that CloudFront forwards to the origin, CloudFront adds one or both of the following:

- An `If-Match` or `If-None-Match` header that contains the `ETag` value for the expired version of the object.
- An `If-Modified-Since` header that contains the `LastModified` value for the expired version of the object.

The origin uses this information to determine whether the object has been updated and, therefore, whether to return the entire object to CloudFront or to return only an HTTP 304 status code (not modified).

> **Note**
>
> `If-Modified-Since` and `If-None-Match` conditional requests are not supported when CloudFront is configured to forward cookies (all or a subset).
> For more information, see Cache content based on cookies.

Cookies

You can configure CloudFront to forward cookies to your origin. For more information, see Cache content based on cookies.

Cross-origin resource sharing (CORS)

If you want CloudFront to respect cross-origin resource sharing settings, configure CloudFront to forward the `Origin` header to your origin. For more information, see Cache content based on request headers.

Encryption

You can require viewers to use HTTPS to send requests to CloudFront and require CloudFront to forward requests to your custom origin by using the protocol that is used by the viewer. For more information, see the following distribution settings:

- Viewer protocol policy
- Protocol (custom origins only)

CloudFront forwards HTTPS requests to the origin server using the SSLv3, TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3 protocols. For custom origins, you can choose the SSL protocols that you want CloudFront to use when communicating with your origin:

- If you're using the CloudFront console, choose protocols by using the **Origin SSL Protocols** check boxes. For more information, see Create a distribution.
- If you're using the CloudFront API, specify protocols by using the `OriginSslProtocols` element. For more information, see OriginSslProtocols and DistributionConfig in the *Amazon CloudFront API Reference*.

If the origin is an Amazon S3 bucket, CloudFront will default to TLSv1.3.

> **Important**
>
> Other versions of SSL and TLS are not supported.

For more information about using HTTPS with CloudFront, see Use HTTPS with CloudFront. For lists of the ciphers that CloudFront supports for HTTPS communication between viewers and CloudFront, and between CloudFront and your origin, see Supported protocols and ciphers between viewers and CloudFront.

GET requests that include a body

If a viewer `GET` request includes a body, CloudFront returns an HTTP status code 403 (Forbidden) to the viewer.

HTTP methods

If you configure CloudFront to process all of the HTTP methods that it supports, CloudFront accepts the following requests from viewers and forwards them to your custom origin:

- `DELETE`
- `GET`
- `HEAD`
- `OPTIONS`
- `PATCH`

- POST
- PUT

CloudFront always caches responses to GET and HEAD requests. You can also configure CloudFront to cache responses to OPTIONS requests. CloudFront does not cache responses to requests that use the other methods.

For information about configuring whether your custom origin processes these methods, see the documentation for your origin.

> **Important**
>
> If you configure CloudFront to accept and forward to your origin all of the HTTP methods that CloudFront supports, configure your origin server to handle all methods. For example, if you configure CloudFront to accept and forward these methods because you want to use POST, you must configure your origin server to handle DELETE requests appropriately so viewers can't delete resources that you don't want them to. For more information, see the documentation for your HTTP server.

HTTP request headers and CloudFront behavior (custom and Amazon S3 origins)

The following table lists HTTP request headers that you can forward to both custom and Amazon S3 origins (with the exceptions that are noted). For each header, the table includes information about the following:

- CloudFront behavior if you don't configure CloudFront to forward the header to your origin, which causes CloudFront to cache your objects based on header values.
- Whether you can configure CloudFront to cache objects based on header values for that header.

  You can configure CloudFront to cache objects based on values in the Date and User-Agent headers, but we don't recommend it. These headers have many possible values, and caching based on their values would cause CloudFront to forward significantly more requests to your origin.

For more information about caching based on header values, see Cache content based on request headers.

Header	Behavior if you don't configure CloudFront to cache based on header values	Caching based on header values is supported
Other-defined headers	**Legacy cache settings** – CloudFront forwards the headers to your origin.	Yes
`Accept`	CloudFront removes the header.	Yes
`Accept-Charset`	CloudFront removes the header.	Yes
`Accept-Encoding`	If the value contains `gzip` or `br`, CloudFront forwards a normalized `Accept-Encoding` header to your origin. For more information, see Compression support and Serve compressed files.	Yes
`Accept-Language`	CloudFront removes the header.	Yes
`Authorization`	• `GET` and `HEAD` requests – CloudFront removes the `Authorization` header field before forwarding the request to your origin. • `OPTIONS` requests – CloudFront removes the `Authorization` header field before forwarding the request to your origin if you configure CloudFront to cache responses to `OPTIONS` requests. CloudFront forwards the `Authorization` header field to your origin if you do not	Yes

Header	Behavior if you don't configure CloudFront to cache based on header values	Caching based on header values is supported
	configure CloudFront to cache responses to OPTIONS requests. • DELETE, PATCH, POST, and PUT requests – CloudFront does not remove the header field before forwarding the request to your origin.	
Cache-Control	CloudFront forwards the header to your origin.	No
CloudFront-Forwarded-Proto	CloudFront does not add the header before forwarding the request to your origin. For more information, see Configure caching based on the protocol of the request.	Yes
CloudFront-Is-Desktop-Viewer	CloudFront does not add the header before forwarding the request to your origin. For more information, see Configure caching based on the device type.	Yes
CloudFront-Is-Mobile-Viewer	CloudFront does not add the header before forwarding the request to your origin. For more information, see Configure caching based on the device type.	Yes

Header	Behavior if you don't configure CloudFront to cache based on header values	Caching based on header values is supported
`CloudFront-Is-Tablet-Viewer`	CloudFront does not add the header before forwarding the request to your origin. For more information, see Configure caching based on the device type.	Yes
`CloudFront-Viewer-Country`	CloudFront does not add the header before forwarding the request to your origin.	Yes
`Connection`	CloudFront replaces this header with `Connection: Keep-Alive` before forwarding the request to your origin.	No
`Content-Length`	CloudFront forwards the header to your origin.	No
`Content-MD5`	CloudFront forwards the header to your origin.	Yes
`Content-Type`	CloudFront forwards the header to your origin.	Yes
`Cookie`	If you configure CloudFront to forward cookies, it will forward the `Cookie` header field to your origin. If you don't, CloudFront removes the `Cookie` header field. For more information, see Cache content based on cookies.	No

Header	Behavior if you don't configure CloudFront to cache based on header values	Caching based on header values is supported
`Date`	CloudFront forwards the header to your origin.	Yes, but not recommended
`Expect`	CloudFront removes the header.	Yes
`From`	CloudFront forwards the header to your origin.	Yes
`Host`	CloudFront sets the value to the domain name of the origin that is associated with the requested object. You can't cache based on the Host header for Amazon S3 or MediaStore origins.	Yes (custom) No (S3 and MediaStore)
`If-Match`	CloudFront forwards the header to your origin.	Yes
`If-Modified-Since`	CloudFront forwards the header to your origin.	Yes
`If-None-Match`	CloudFront forwards the header to your origin.	Yes
`If-Range`	CloudFront forwards the header to your origin.	Yes
`If-Unmodified-Since`	CloudFront forwards the header to your origin.	Yes

Header	Behavior if you don't configure CloudFront to cache based on header values	Caching based on header values is supported
`Max-Forwards`	CloudFront forwards the header to your origin.	No
`Origin`	CloudFront forwards the header to your origin.	Yes
`Pragma`	CloudFront forwards the header to your origin.	No
`Proxy-Authenticate`	CloudFront removes the header.	No
`Proxy-Authorization`	CloudFront removes the header.	No
`Proxy-Connection`	CloudFront removes the header.	No
`Range`	CloudFront forwards the header to your origin. For more information, see How CloudFront processes partial requests for an object (range GETs).	Yes, by default
`Referer`	CloudFront removes the header.	Yes
`Request-Range`	CloudFront forwards the header to your origin.	No
`TE`	CloudFront removes the header.	No
`Trailer`	CloudFront removes the header.	No

Header	Behavior if you don't configure CloudFront to cache based on header values	Caching based on header values is supported
`Transfer-Encoding`	CloudFront forwards the header to your origin.	No
`Upgrade`	CloudFront removes the header, unless you've established a WebSocket connection.	No (except for WebSocket connections)
`User-Agent`	CloudFront replaces the value of this header field with `Amazon CloudFront`. If you want CloudFront to cache your content based on the device the user is using, see Configure caching based on the device type.	Yes, but not recommended
`Via`	CloudFront forwards the header to your origin.	Yes
`Warning`	CloudFront forwards the header to your origin.	Yes
`X-Amz-Cf-Id`	CloudFront adds the header to the viewer request before forwarding the request to your origin. The header value contains an encrypted string that uniquely identifies the request.	No
`X-Edge-*`	CloudFront removes all `X-Edge-*` headers.	No
`X-Forwarded-For`	CloudFront forwards the header to your origin. For more information, see Client IP addresses.	Yes

Header	Behavior if you don't configure CloudFront to cache based on header values	Caching based on header values is supported
`X-Forwarded-Proto`	CloudFront removes the header.	No
`X-HTTP-Method-Override`	CloudFront removes the header.	Yes
`X-Real-IP`	CloudFront removes the header.	No

HTTP version

By default, CloudFront forwards requests to your custom origin using HTTP/1.1. To have CloudFront use HTTP/2 when your origin supports it, see Origin HTTP version (custom and VPC origins only).

Maximum length of a request and maximum length of a URL

The maximum length of a request, including the path, the query string (if any), and headers, is 32,768 bytes.

CloudFront constructs a URL from the request. The maximum length of this URL is 8192 bytes.

If a URL exceeds the maximum length, CloudFront returns HTTP status code 414 (URI Too Long) to the viewer. If a request exceeds the maximum length because the header size is exceeded, CloudFront returns HTTP status code 494 to the viewer. In both cases, CloudFront then terminates the TCP connection to the viewer.

OCSP stapling

When a viewer submits an HTTPS request for an object, either CloudFront or the viewer must confirm with the certificate authority (CA) that the SSL certificate for the domain has not been revoked. OCSP stapling speeds up certificate validation by allowing CloudFront to validate the

certificate and to cache the response from the CA, so the client doesn't need to validate the certificate directly with the CA.

The performance improvement of OCSP stapling is more pronounced when CloudFront receives numerous HTTPS requests for objects in the same domain. Each server in a CloudFront edge location must submit a separate validation request. When CloudFront receives a lot of HTTPS requests for the same domain, every server in the edge location soon has a response from the CA that it can "staple" to a packet in the SSL handshake; when the viewer is satisfied that the certificate is valid, CloudFront can serve the requested object. If your distribution doesn't get much traffic in a CloudFront edge location, new requests are more likely to be directed to a server that hasn't validated the certificate with the CA yet. In that case, the viewer separately performs the validation step and the CloudFront server serves the object. That CloudFront server also submits a validation request to the CA, so the next time it receives a request that includes the same domain name, it has a validation response from the CA.

Persistent connections

When CloudFront gets a response from your origin, it tries to maintain the connection for several seconds in case another request arrives during that period. Maintaining a persistent connection saves the time required to re-establish the TCP connection and perform another TLS handshake for subsequent requests.

For more information, including how to configure the duration of persistent connections, see Keep-alive timeout (custom and VPC origins only) in the section All distribution settings reference.

Protocols

CloudFront forwards HTTP or HTTPS requests to the origin server based on the following:

- The protocol of the request that the viewer sends to CloudFront, either HTTP or HTTPS.
- The value of the **Origin Protocol Policy** field in the CloudFront console or, if you're using the CloudFront API, the `OriginProtocolPolicy` element in the `DistributionConfig` complex type. In the CloudFront console, the options are **HTTP Only**, **HTTPS Only**, and **Match Viewer**.

If you specify **HTTP Only** or **HTTPS Only**, CloudFront forwards requests to the origin server using the specified protocol, regardless of the protocol in the viewer request.

If you specify **Match Viewer**, CloudFront forwards requests to the origin server using the protocol in the viewer request. Note that CloudFront caches the object only once even if viewers make requests using both HTTP and HTTPS protocols.

> **Important**
>
> If CloudFront forwards a request to the origin using the HTTPS protocol, and if the origin server returns an invalid certificate or a self-signed certificate, CloudFront drops the TCP connection.

For information about how to update a distribution using the CloudFront console, see Update a distribution. For information about how to update a distribution using the CloudFront API, go to UpdateDistribution in the *Amazon CloudFront API Reference*.

Query strings

You can configure whether CloudFront forwards query string parameters to your origin. For more information, see Cache content based on query string parameters.

Origin connection timeout and attempts

Origin connection timeout is the number of seconds that CloudFront waits when trying to establish a connection to the origin.

Origin connection attempts is the number of times that CloudFront attempts to connect to the origin.

Together, these settings determine how long CloudFront tries to connect to the origin before failing over to the secondary origin (in the case of an origin group) or returning an error response to the viewer. By default, CloudFront waits as long as 30 seconds (3 attempts of 10 seconds each) before attempting to connect to the secondary origin or returning an error response. You can reduce this time by specifying a shorter connection timeout, fewer attempts, or both.

For more information, see Control origin timeouts and attempts.

Origin response timeout

The *origin response timeout*, also known as the *origin read timeout* or *origin request timeout*, applies to both of the following:

- The amount of time, in seconds, that CloudFront waits for a response after forwarding a request to the origin.
- The amount of time, in seconds, that CloudFront waits after receiving a packet of a response from the origin and before receiving the next packet.

CloudFront behavior depends on the HTTP method of the viewer request:

- `GET` and `HEAD` requests – If the origin doesn't respond or stops responding within the duration of the response timeout, CloudFront drops the connection. If the specified number of origin connection attempts is more than 1, CloudFront tries again to get a complete response. CloudFront tries up to 3 times, as determined by the value of the *origin connection attempts* setting. If the origin doesn't respond during the final attempt, CloudFront doesn't try again until it receives another request for content on the same origin.
- `DELETE`, `OPTIONS`, `PATCH`, `PUT`, and `POST` requests – If the origin doesn't respond for the duration of the read timeout, CloudFront drops the connection and doesn't try again to contact the origin. The client can resubmit the request if necessary.

For more information, including how to configure the origin response timeout, see Response timeout.

Simultaneous requests for the same object (request collapsing)

When a CloudFront edge location receives a request for an object and the object isn't in the cache or the cached object is expired, CloudFront immediately sends the request to the origin. However, if there are simultaneous requests for the same object—that is, if additional requests for the same object (with the same cache key) arrive at the edge location before CloudFront receives the response to the first request—CloudFront pauses before forwarding the additional requests to the origin. This brief pause helps to reduce the load on the origin. CloudFront sends the response from the original request to all the requests that it received while it was paused. This is called *request collapsing*. In CloudFront logs, the first request is identified as a `Miss` in the `x-edge-result-type` field, and the collapsed requests are identified as a `Hit`. For more information about CloudFront logs, see the section called "CloudFront and edge function logging".

CloudFront only collapses requests that share a *cache key*. If the additional requests do not share the same cache key because, for example, you configured CloudFront to cache based on request

headers or cookies or query strings, CloudFront forwards all the requests with a unique cache key to your origin.

If you want to prevent all request collapsing, you can use the managed cache policy `CachingDisabled`, which also prevents caching. For more information, see Use managed cache policies.

If you want to prevent request collapsing for specific objects, you can set the minimum TTL for the cache behavior to 0 *and* configure the origin to send `Cache-Control: private`, `Cache-Control: no-store`, `Cache-Control: no-cache`, `Cache-Control: max-age=0`, or `Cache-Control: s-maxage=0`. These configurations will increase the load on your origin and introduce additional latency for the simultaneous requests that are paused while CloudFront waits for the response to the first request.

`User-Agent` header

If you want CloudFront to cache different versions of your objects based on the device that a user is using to view your content, we recommend that you configure CloudFront to forward one or more of the following headers to your custom origin:

- `CloudFront-Is-Desktop-Viewer`
- `CloudFront-Is-Mobile-Viewer`
- `CloudFront-Is-SmartTV-Viewer`
- `CloudFront-Is-Tablet-Viewer`

Based on the value of the `User-Agent` header, CloudFront sets the value of these headers to `true` or `false` before forwarding the request to your origin. If a device falls into more than one category, more than one value might be `true`. For example, for some tablet devices, CloudFront might set both `CloudFront-Is-Mobile-Viewer` and `CloudFront-Is-Tablet-Viewer` to `true`. For more information about configuring CloudFront to cache based on request headers, see Cache content based on request headers.

You can configure CloudFront to cache objects based on values in the `User-Agent` header, but we don't recommend it. The `User-Agent` header has many possible values, and caching based on those values would cause CloudFront to forward significantly more requests to your origin.

If you do not configure CloudFront to cache objects based on values in the `User-Agent` header, CloudFront adds a `User-Agent` header with the following value before it forwards a request to your origin:

```
User-Agent = Amazon CloudFront
```

CloudFront adds this header regardless of whether the request from the viewer includes a `User-Agent` header. If the request from the viewer includes a `User-Agent` header, CloudFront removes it.

How CloudFront processes responses from your custom origin

Learn how CloudFront processes responses from your custom origin.

Contents

- 100 Continue responses
- Caching
- Canceled requests
- Content negotiation
- Cookies
- Dropped TCP connections
- HTTP response headers that CloudFront removes or replaces
- Maximum cacheable file size
- Origin unavailable
- Redirects
- Transfer-Encoding header

`100 Continue` responses

Your origin cannot send more than one 100-Continue response to CloudFront. After the first 100-Continue response, CloudFront expects an HTTP 200 OK response. If your origin sends another 100-Continue response after the first one, CloudFront will return an error.

Caching

- Make sure that the origin server sets valid and accurate values for the `Date` and `Last-Modified` header fields.

- CloudFront normally respects a `Cache-Control: no-cache` header in the response from the origin. For an exception, see Simultaneous requests for the same object (request collapsing).

Canceled requests

If an object is not in the edge cache, and if a viewer terminates a session (for example, closes a browser) after CloudFront gets the object from your origin but before it can deliver the requested object, CloudFront does not cache the object in the edge location.

Content negotiation

If your origin returns `Vary:*` in the response, and if the value of **Minimum TTL** for the corresponding cache behavior is **0**, CloudFront caches the object but still forwards every subsequent request for the object to the origin to confirm that the cache contains the latest version of the object. CloudFront doesn't include any conditional headers, such as `If-None-Match` or `If-Modified-Since`. As a result, your origin returns the object to CloudFront in response to every request.

If your origin returns `Vary:*` in the response, and if the value of **Minimum TTL** for the corresponding cache behavior is any other value, CloudFront processes the `Vary` header as described in HTTP response headers that CloudFront removes or replaces.

Cookies

If you enable cookies for a cache behavior, and if the origin returns cookies with an object, CloudFront caches both the object and the cookies. Note that this reduces cacheability for an object. For more information, see Cache content based on cookies.

Dropped TCP connections

If the TCP connection between CloudFront and your origin drops while your origin is returning an object to CloudFront, CloudFront behavior depends on whether your origin included a `Content-Length` header in the response:

- **Content-Length header** – CloudFront returns the object to the viewer as it gets the object from your origin. However, if the value of the `Content-Length` header doesn't match the size of the object, CloudFront doesn't cache the object.

- **Transfer-Encoding: Chunked** – CloudFront returns the object to the viewer as it gets the object from your origin. However, if the chunked response is not complete, CloudFront does not cache the object.
- **No Content-Length header** – CloudFront returns the object to the viewer and caches it, but the object might not be complete. Without a `Content-Length` header, CloudFront cannot determine whether the TCP connection was dropped accidentally or on purpose.

We recommend that you configure your HTTP server to add a `Content-Length` header to prevent CloudFront from caching partial objects.

HTTP response headers that CloudFront removes or replaces

CloudFront removes or updates the following header fields before forwarding the response from your origin to the viewer:

- `Set-Cookie` – If you configure CloudFront to forward cookies, it will forward the `Set-Cookie` header field to clients. For more information, see Cache content based on cookies.
- `Trailer`
- `Transfer-Encoding` – If your origin returns this header field, CloudFront sets the value to `chunked` before returning the response to the viewer.
- `Upgrade`
- `Vary` – Note the following:
  - If you configure CloudFront to forward any of the device-specific headers to your origin (`CloudFront-Is-Desktop-Viewer`, `CloudFront-Is-Mobile-Viewer`, `CloudFront-Is-SmartTV-Viewer`, `CloudFront-Is-Tablet-Viewer`) and you configure your origin to return `Vary:User-Agent` to CloudFront, CloudFront returns `Vary:User-Agent` to the viewer. For more information, see Configure caching based on the device type.
  - If you configure your origin to include either `Accept-Encoding` or `Cookie` in the `Vary` header, CloudFront includes the values in the response to the viewer.
  - If you configure CloudFront to forward headers to your origin, and if you configure your origin to return the header names to CloudFront in the `Vary` header (for example, `Vary:Accept-Charset,Accept-Language`), CloudFront returns the `Vary` header with those values to the viewer.
  - For information about how CloudFront processes a value of `*` in the `Vary` header, see Content negotiation.

- If you configure your origin to include any other values in the `Vary` header, CloudFront removes the values before returning the response to the viewer.

- `Via` – CloudFront sets the value to the following in the response to the viewer:

  `Via: http-version alphanumeric-string.cloudfront.net (CloudFront)`

  For example, the value is something like the following:

  `Via: 1.1 1026589cc7887e7a0dc7827b4example.cloudfront.net (CloudFront)`

Maximum cacheable file size

The maximum size of a response body that CloudFront saves in its cache is 50 GB. This includes chunked transfer responses that don't specify the `Content-Length` header value.

You can use CloudFront to cache an object that is larger than this size by using range requests to request the objects in parts that are each 50 GB or smaller. CloudFront caches these parts because each of them is 50 GB or smaller. After the viewer retrieves all the parts of the object, it can reconstruct the original, larger object. For more information, see Use range requests to cache large objects.

Origin unavailable

If your origin server is unavailable and CloudFront gets a request for an object that is in the edge cache but that has expired (for example, because the period of time specified in the `Cache-Control max-age` directive has passed), CloudFront either serves the expired version of the object or serves a custom error page. For more information about CloudFront behavior when you've configured custom error pages, see How CloudFront processes errors when you have configured custom error pages.

In some cases, an object that is seldom requested is evicted and is no longer available in the edge cache. CloudFront can't serve an object that has been evicted.

Redirects

If you change the location of an object on the origin server, you can configure your web server to redirect requests to the new location. After you configure the redirect, the first time a viewer submits a request for the object, CloudFront sends the request to the origin, and the origin

responds with a redirect (for example, `302 Moved Temporarily`). CloudFront caches the redirect and returns it to the viewer. CloudFront does not follow the redirect.

You can configure your web server to redirect requests to one of the following locations:

- The new URL of the object on the origin server. When the viewer follows the redirect to the new URL, the viewer bypasses CloudFront and goes straight to the origin. As a result, we recommend that you don't redirect requests to the new URL of the object on the origin.
- The new CloudFront URL for the object. When the viewer submits the request that contains the new CloudFront URL, CloudFront gets the object from the new location on your origin, caches it at the edge location, and returns the object to the viewer. Subsequent requests for the object will be served by the edge location. This avoids the latency and load associated with viewers requesting the object from the origin. However, every new request for the object will incur charges for two requests to CloudFront.

`Transfer-Encoding` header

CloudFront supports only the `chunked` value of the `Transfer-Encoding` header. If your origin returns `Transfer-Encoding: chunked`, CloudFront returns the object to the client as the object is received at the edge location, and caches the object in chunked format for subsequent requests.

If the viewer makes a `Range GET` request and the origin returns `Transfer-Encoding: chunked`, CloudFront returns the entire object to the viewer instead of the requested range.

We recommend that you use chunked encoding if the content length of your response cannot be predetermined. For more information, see Dropped TCP connections.

Request and response behavior for origin groups

Requests to an origin group work the same as requests to an origin that is not set up as an origin group, except when there is an origin failover. As with any other origin, when CloudFront receives a request and the content is already cached in an edge location, the content is served to viewers from the cache. When there's a cache miss and the origin is an origin group, viewer requests are forwarded to the primary origin in the origin group.

The request and response behavior for the primary origin is the same as it is for an origin that isn't in an origin group. For more information, see Request and response behavior for Amazon S3 origins and Request and response behavior for custom origins.

The following describes the behavior for origin failover when the primary origin returns specific HTTP status codes:

- HTTP 2xx status code (success): CloudFront caches the file and returns it to the viewer.
- HTTP 3xx status code (redirection): CloudFront returns the status code to the viewer.
- HTTP 4xx or 5xx status code (client/server error): If the returned status code has been configured for failover, CloudFront sends the same request to the secondary origin in the origin group.
- HTTP 4xx or 5xx status code (client/server error): If the returned status code has not been configured for failover, CloudFront returns the error to the viewer.

CloudFront fails over to the secondary origin only when the HTTP method of the viewer request is `GET`, `HEAD`, or `OPTIONS`. CloudFront does not fail over when the viewer sends a different HTTP method (for example `POST`, `PUT`, and so on).

When CloudFront sends a request to a secondary origin, the response behavior is the same as for a CloudFront origin that's not in an origin group.

For more information about origin groups, see Optimize high availability with CloudFront origin failover.

Add custom headers to origin requests

You can configure CloudFront to add custom headers to the requests that it sends to your origin. You can use custom headers to send and gather information from your origin that you don't get with typical viewer requests. You can even customize the headers for each origin. CloudFront supports custom headers for custom origins and Amazon S3 origins.

Contents

- Use cases
- Configure CloudFront to add custom headers to origin requests
- Custom headers that CloudFront can't add to origin requests
- Configure CloudFront to forward the Authorization header

Use cases

You can use custom headers, such as the following examples:

Identifying requests from CloudFront

You can identify the requests that your origin receives from CloudFront. This can be useful if you want to know if users are bypassing CloudFront, or if you're using more than one CDN and you want information about which requests are coming from each CDN.

> **Note**
>
> If you're using an Amazon S3 origin and you enable Amazon S3 server access logging, the logs don't include header information.

Determining which requests come from a particular distribution

If you configure more than one CloudFront distribution to use the same origin, you can add different custom headers in each distribution. You can then use the logs from your origin to determine which requests came from which CloudFront distribution.

Enabling cross-origin resource sharing (CORS)

If some of your viewers don't support cross-origin resource sharing (CORS), you can configure CloudFront to always add the `Origin` header to requests that it sends to your origin. Then you can configure your origin to return the `Access-Control-Allow-Origin` header for every request. You must also configure CloudFront to respect CORS settings.

Controlling access to content

You can use custom headers to control access to content. By configuring your origin to respond to requests only when they include a custom header that gets added by CloudFront, you prevent users from bypassing CloudFront and accessing your content directly on the origin. For more information, see Restrict access to files on custom origins.

Configure CloudFront to add custom headers to origin requests

To configure a distribution to add custom headers to requests that it sends to your origin, update the origin configuration using one of the following methods:

- **CloudFront console** – When you create or update a distribution, specify header names and values in the **Add custom headers** settings. For more information, see Add custom header.

- **CloudFront API** – For each origin that you want to add custom headers to, specify the header names and values in the `CustomHeaders` field inside `Origin`. For more information, see CreateDistribution or UpdateDistribution in the *Amazon CloudFront API Reference*.

If the header names and values that you specify are not already present in the viewer request, CloudFront adds them to the origin request. If a header is present, CloudFront overwrites the header value before forwarding the request to the origin.

For the quotas that apply to origin custom headers, see Quotas on headers.

Custom headers that CloudFront can't add to origin requests

You can't configure CloudFront to add any of the following headers to requests that it sends to your origin:

- `Cache-Control`
- `Connection`
- `Content-Length`
- `Cookie`
- `Host`
- `If-Match`
- `If-Modified-Since`
- `If-None-Match`
- `If-Range`
- `If-Unmodified-Since`
- `Max-Forwards`
- `Pragma`
- `Proxy-Authenticate`
- `Proxy-Authorization`
- `Proxy-Connection`
- `Range`
- `Request-Range`

- `TE`
- `Trailer`
- `Transfer-Encoding`
- `Upgrade`
- `Via`
- Headers that begin with `X-Amz-`
- Headers that begin with `X-Edge-`
- `X-Real-Ip`

Configure CloudFront to forward the `Authorization` header

When CloudFront forwards a viewer request to your origin, CloudFront removes some viewer headers by default, including the `Authorization` header. To make sure that your origin always receives the `Authorization` header in origin requests, you have the following options:

- Add the `Authorization` header to the cache key using a cache policy. All headers in the cache key are automatically included in origin requests. For more information, see Control the cache key with a policy.
- Add the `Authorization` header individually in an origin request policy. For more information, see Control origin requests with a policy.
- Use an origin request policy that forwards all viewer headers to the origin. CloudFront provides a managed origin request policy for this use case, called **Managed-AllViewer**. For more information, see Use managed origin request policies.

> **Important**
>
> If you forward the `Authorization` header to your origin without including it in the cache key, make sure that your origin does not rely on the `Authorization` header for access control of cached content. When the `Authorization` header is not part of the cache key, CloudFront can serve the same cached response to both authorized and unauthorized viewers. Either include the `Authorization` header in the cache key using a cache policy, or disable caching entirely for origins that require origin-side authorization processing.

How CloudFront processes partial requests for an object (range GETs)

For a large object, the viewer (web browser or other client) can make multiple GET requests and use the Range request header to download the object in smaller parts. These requests for ranges of bytes, sometimes known as Range GET requests, improve the efficiency of partial downloads and the recovery from partially failed transfers.

When CloudFront receives a Range GET request, it checks the cache in the edge location that received the request. If the cache in that edge location already contains the entire object or the requested part of the object, CloudFront immediately serves the requested range from the cache.

If the cache doesn't contain the requested range, CloudFront forwards the request to the origin. (To optimize performance, CloudFront might request a larger range than the client requested in the Range GET.) What happens next depends on whether the origin supports Range GET requests:

- **If the origin supports Range GET requests** – It returns the requested range. CloudFront serves the requested range and also caches it for future requests. (Amazon S3 supports Range GET requests, as do many HTTP servers.)
- **If the origin doesn't support Range GET requests** – It returns the entire object. CloudFront serves the current request by sending the entire object while also caching it for future requests. After CloudFront caches the entire object in an edge cache, it responds to new Range GET requests by serving the requested range.

In either case, CloudFront begins to serve the requested range or object to the end user as soon as the first byte arrives from the origin.

> **Note**
>
> If the viewer makes a Range GET request and the origin returns `Transfer-Encoding: chunked`, CloudFront returns the entire object to the viewer instead of the requested range.

CloudFront generally follows the RFC specification for the Range header. However, if your Range headers don't adhere to the following requirements, CloudFront returns HTTP status code 200 with the full object instead of status code 206 with the specified ranges:

- The ranges must be listed in ascending order. For example, `100-200,300-400` is valid, `300-400,100-200` is not valid.
- The ranges must not overlap. For example, `100-200,150-250` is not valid.
- All of the range specifications must be valid. For example, you can't specify a negative value as part of a range.

For more information about the Range request header, see Range Requests in RFC 7233, or Range in the MDN Web Docs.

Use range requests to cache large objects

When caching is enabled, CloudFront doesn't retrieve or cache an object that is larger than 50 GB. When an origin indicates that the object is larger than this size (in the `Content-Length` response header), CloudFront closes the connection to the origin and returns an error to the viewer. (With caching disabled, CloudFront can retrieve an object that is larger than this size from the origin and pass it along to the viewer. However, CloudFront doesn't cache the object.)

However, with range requests, you can use CloudFront to cache an object that is larger than the maximum cacheable file size.

Example

1. Consider an origin with a 100 GB object. With caching enabled, CloudFront doesn't retrieve or cache an object this large. However, the viewer can send multiple range requests to retrieve this object in parts, with each part smaller than 50 GB.
2. The viewer can request the object in 20 GB parts by sending a request with the header `Range: bytes=0-21474836480` to retrieve the first part, another request with the header `Range: bytes=21474836481-42949672960` to retrieve the next part, and so on.
3. When the viewer has received all of the parts, it can combine them to construct the original 100 GB object.
4. In this case, CloudFront caches each of the 20 GB parts of the object and can respond to subsequent requests for the same part from the cache.

For a range request for a compressed object, the byte range request is based on the compressed size, and not the original size of the object. For more information about compressing files, see Serve compressed files.

How CloudFront processes HTTP 3xx status codes from your origin

When CloudFront requests an object from your Amazon S3 bucket or custom origin server, your origin sometimes returns an HTTP 3xx status code. This typically indicates one of the following:

- The object's URL has changed (for example, status codes 301, 302, 307, or 308)
- The object hasn't changed since the last time CloudFront requested it (status code 304)

CloudFront caches 3xx responses according to the settings in your CloudFront distribution and the headers in the response. CloudFront caches 307 and 308 responses only when you include the `Cache-Control` header in responses from the origin. For more information, see Manage how long content stays in the cache (expiration).

If your origin returns a redirect status code (for example, 301 or 307), CloudFront doesn't follow the redirect. CloudFront passes along the 301 or 307 response to the viewer, who can follow the redirect by sending a new request.

How CloudFront processes HTTP 4xx and 5xx status codes from your origin

When CloudFront requests an object from your Amazon S3 bucket or custom origin server, your origin sometimes returns an HTTP 4xx or 5xx status code, which indicates that an error has occurred. CloudFront behavior depends on:

- Whether you have configured custom error pages
- Whether you have configured how long you want CloudFront to cache error responses from your origin (error caching minimum TTL)
- The status code
- For 5xx status codes, whether the requested object is currently in the CloudFront edge cache
- For some 4xx status codes, whether the origin returns a `Cache-Control max-age` or `Cache-Control s-maxage` header

CloudFront always caches responses to `GET` and `HEAD` requests. You can also configure CloudFront to cache responses to `OPTIONS` requests. CloudFront does not cache responses to requests that use the other methods.

If the origin doesn't respond, the CloudFront request to the origin times out which is considered an HTTP 5xx error from the origin, even though the origin didn't respond with that error. In that scenario, CloudFront continues to serve cached content. For more information, see Origin unavailable.

If you have enabled logging, CloudFront writes the results to the logs regardless of the HTTP status code.

For more information about features and options that relate to the error message returned from CloudFront, see the following:

- For information about settings for custom error pages in the CloudFront console, see Custom error pages and error caching.
- For information about the error caching minimum TTL in the CloudFront console, see Error caching minimum TTL (seconds).
- For a list of the HTTP status codes that CloudFront caches, see HTTP 4xx and 5xx status codes that CloudFront caches.

Topics

- How CloudFront processes errors when you have configured custom error pages
- How CloudFront processes errors if you haven't configured custom error pages
- HTTP 4xx and 5xx status codes that CloudFront caches

How CloudFront processes errors when you have configured custom error pages

If you have configured custom error pages, CloudFront behavior depends on whether the requested object is in the edge cache.

The requested object is not in the edge cache

CloudFront continues to try to get the requested object from your origin when all of the following are true:

- A viewer requests an object.
- The object isn't in the edge cache.
- Your origin returns an HTTP 4xx or 5xx status code and one of the following is true:
  - Your origin returns an HTTP 5xx status code instead of returning a 304 status code (Not Modified) or an updated version of the object.
  - Your origin returns an HTTP 4xx status code that is not restricted by a cache control header and is included in the following list of status codes: HTTP 4xx and 5xx status codes that CloudFront caches.
  - Your origin returns an HTTP 4xx status code with a `Cache-Control max-age` header or a `Cache-Control s-maxage` header, and the status code is included in the following list of status codes: Control HTTP 4xx status codes that CloudFront caches based on `Cache-Control` headers.

CloudFront does the following:

1. In the CloudFront edge cache that received the viewer request, CloudFront checks your distribution configuration and gets the path of the custom error page that corresponds with the status code that your origin returned.
2. CloudFront finds the first cache behavior in your distribution that has a path pattern that matches the path of the custom error page.
3. The CloudFront edge location sends a request for the custom error page to the origin that is specified in the cache behavior.
4. The origin returns the custom error page to the edge location.
5. CloudFront returns the custom error page to the viewer that made the request, and also caches the custom error page for the maximum of the following:
   - The amount of time specified by the error caching minimum TTL (10 seconds by default)
   - The amount of time specified by a `Cache-Control max-age` header or a `Cache-Control s-maxage` header that is returned by the origin when the first request generated the error
6. After the caching time (determined in Step 5) has elapsed, CloudFront tries again to get the requested object by forwarding another request to your origin. CloudFront continues to retry at intervals specified by the error caching minimum TTL.

Note

If you also configured a cache behavior for the same custom error page, CloudFront uses the cache behavior TTL instead. In this case, CloudFront will do the following for steps 5 and 6:

- After CloudFront returns the custom error page to the viewer that made the request, CloudFront checks the cache behavior TTL (for example, you set the default TTL to 5 seconds). CloudFront then caches the custom error page up to that maximum.
- After 5 seconds elapse, CloudFront fetches the custom error page from the origin again. CloudFront will continue to retry at intervals specified by the cache behavior TTL.

For more information, see the cache behavior TTL settings.

The requested object is in the edge cache

CloudFront continues to serve the object that is currently in the edge cache when all of the following are true:

- A viewer requests an object.
- The object is in the edge cache but it has expired.
- Your origin returns an HTTP 5xx status code instead of returning a 304 status code (Not Modified) or an updated version of the object.

CloudFront does the following:

1. If your origin returns a 5xx status code, CloudFront serves the object even though it has expired. For the duration of the error caching minimum TTL, CloudFront continues to respond to viewer requests by serving the object from the edge cache.

   If your origin returns a 4xx status code, CloudFront returns the status code, not the requested object, to the viewer.
2. After the error caching minimum TTL has elapsed, CloudFront tries again to get the requested object by forwarding another request to your origin. Note that if the object is not requested frequently, CloudFront might evict it from the edge cache while your origin server is still

returning 5xx responses. For information about how long objects stay in CloudFront edge caches, see Manage how long content stays in the cache (expiration).

How CloudFront processes errors if you haven't configured custom error pages

If you haven't configured custom error pages, CloudFront behavior depends on whether the requested object is in the edge cache.

Topics

- The requested object is not in the edge cache
- The requested object is in the edge cache

The requested object is not in the edge cache

CloudFront continues to try to get the requested object from your origin when all of the following are true:

- A viewer requests an object.
- The object isn't in the edge cache.
- Your origin returns an HTTP 4xx or 5xx status code and one of the following is true:
  - Your origin returns an HTTP 5xx status code instead of returning a 304 status code (Not Modified) or an updated version of the object.
  - Your origin returns an HTTP 4xx status code that is not restricted by a cache control header and is included in the following list of status codes: HTTP 4xx and 5xx status codes that CloudFront caches
  - Your origin returns an HTTP 4xx status code with a `Cache-Control max-age` header or a `Cache-Control s-maxage` header and the status code is included in the following list of status codes: Control HTTP 4xx status codes that CloudFront caches based on `Cache-Control` headers.

CloudFront does the following:

1. CloudFront returns the 4xx or 5xx status code to the viewer, and also caches status code in the edge cache that received the request for the maximum of the following:

- The amount of time specified by the error caching minimum TTL (10 seconds by default)
- The amount of time specified by a `Cache-Control max-age` header or a `Cache-Control s-maxage` header that is returned by the origin when the first request generated the error

2. For the duration of the caching time (determined in Step 1), CloudFront responds to subsequent viewer requests for the same object with the cached 4xx or 5xx status code.
3. After the caching time (determined in Step 1) has elapsed, CloudFront tries again to get the requested object by forwarding another request to your origin. CloudFront continues to retry at intervals specified by the error caching minimum TTL.

The requested object is in the edge cache

CloudFront continues to serve the object that is currently in the edge cache when all of the following are true:

- A viewer requests an object.
- The object is in the edge cache but it has expired. This means the object is *stale*.
- Your origin returns an HTTP 5xx status code instead of returning a 304 status code (Not Modified) or an updated version of the object.

CloudFront does the following:

1. If your origin returns a 5xx error code, CloudFront serves the object even though it has expired. For the duration of the error caching minimum TTL (10 seconds by default), CloudFront continues to respond to viewer requests by serving the object from the edge cache.

   If your origin returns a 4xx status code, CloudFront returns the status code, not the requested object, to the viewer.
2. After the error caching minimum TTL has elapsed, CloudFront tries again to get the requested object by forwarding another request to your origin. If the object isn't requested frequently, CloudFront might evict it from the edge cache while your origin server is still returning 5xx responses. For more information, see Manage how long content stays in the cache (expiration).

Tip

- If you configure the `stale-if-error` or `Stale-While-Revalidate` directive, you can specify how long the stale objects are available in the edge cache. With this, you can continue serving content for your viewers even when your origin is unavailable. For information, see Serve stale (expired) content.
- CloudFront will only serve an object that is stale up to the specified maximum TTL value. After this duration, the object won't be available from the edge cache.

HTTP 4xx and 5xx status codes that CloudFront caches

CloudFront caches HTTP 4xx and 5xx status codes returned by your origin, depending on the specific status code that is returned and whether your origin returns specific headers in the response.

CloudFront caches the following HTTP 4xx and 5xx status codes returned by your origin. If you configured a custom error page for an HTTP status code, CloudFront caches the custom error page.

Note

If you're using the CachingDisabled managed cache policy, CloudFront won't cache these status codes or custom error pages.

HTTP status code	Description
404	Not Found
414	Request-URI Too Large
500	Internal Server Error
501	Not Implemented

HTTP status code	Description
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out

HTTP 4xx status codes that CloudFront caches based on `Cache-Control` headers

CloudFront only caches the following HTTP 4xx status codes returned by your origin if your origin returns a `Cache-Control max-age` or `Cache-Control s-maxage` header. If you have configured a custom error page for one of these HTTP status codes – and your origin returns one of the cache control headers – CloudFront caches the custom error page.

HTTP status code	Description
400	Bad Request
403	Forbidden
405	Method Not Allowed
412[1]	Precondition Failed
415[1]	Unsupported Media Type

[1]CloudFront doesn't support creating custom error pages for these HTTP status codes.

Generate custom error responses

If an object that you're serving through CloudFront is unavailable for some reason, your web server typically returns a relevant HTTP status code to CloudFront to indicate this. For example, if a viewer requests an invalid URL, your web server returns an HTTP 404 (Not Found) status code to CloudFront, and then CloudFront returns that status code to the viewer. Instead of using this default error response, you can create a custom one that CloudFront returns to the viewer.

If you configure CloudFront to return a custom error page for an HTTP status code but the custom error page isn't available, CloudFront returns to the viewer the status code that CloudFront received from the origin that contains the custom error pages. For example, suppose your custom origin returns a 500 status code and you have configured CloudFront to get a custom error page for a 500 status code from an Amazon S3 bucket. However, someone accidentally deleted the custom error page from your Amazon S3 bucket. CloudFront returns an HTTP 404 status code (Not Found) to the viewer that requested the object.

When CloudFront returns a custom error page to a viewer, you pay the standard CloudFront charges for the custom error page, not the charges for the requested object. For more information about CloudFront charges, see Amazon CloudFront Pricing.

Topics

- Configure error response behavior
- Create a custom error page for specific HTTP status codes
- Store objects and custom error pages in different locations
- Change response codes returned by CloudFront
- Control how long CloudFront caches errors

Configure error response behavior

You have several options to manage how CloudFront responds when there's an error. To configure custom error responses, you can use the CloudFront console, the CloudFront API, or CloudFormation. Regardless of how you choose to update the configuration, consider the following tips and recommendations:

- Save your custom error pages in a location that is accessible to CloudFront. We recommend that you store them in an Amazon S3 bucket, and that you don't store them in the same place as the rest of your website or application's content. If you store the custom error pages on the same

origin as your website or application, and the origin starts to return 5xx errors, CloudFront can't get the custom error pages because the origin server is unavailable. For more information, see Store objects and custom error pages in different locations.

- Make sure that CloudFront has permission to get your custom error pages. If the custom error pages are stored in Amazon S3, the pages must be publicly accessible or you must configure a CloudFront origin access control (OAC). If the custom error pages are stored in a custom origin, the pages must be publicly accessible.
- (Optional) Configure your origin to add a `Cache-Control` or `Expires` header along with the custom error pages, if you want. You can also use the **Error Caching Minimum TTL** setting to control how long CloudFront caches the custom error pages. For more information, see Control how long CloudFront caches errors.

Configure custom error responses

To configure custom error responses in the CloudFront console, you must have a CloudFront distribution. In the console, the configuration settings for custom error responses are only available for existing distributions. To learn how to create a distribution, see Get started with a CloudFront standard distribution.

Console

To configure custom error responses (console)

1. Sign in to the AWS Management Console and open the **Distributions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#distributions.
2. In the list of distributions, choose the distribution to update.
3. Choose the **Error Pages** tab, then choose **Create Custom Error Response**.
4. Enter the applicable values. For more information, see Custom error pages and error caching.
5. After entering the desired values, choose **Create**.

CloudFront API or CloudFormation

To configure custom error responses with the CloudFront API or CloudFormation, use the `CustomErrorResponse` type in a distribution. For more information, see the following:

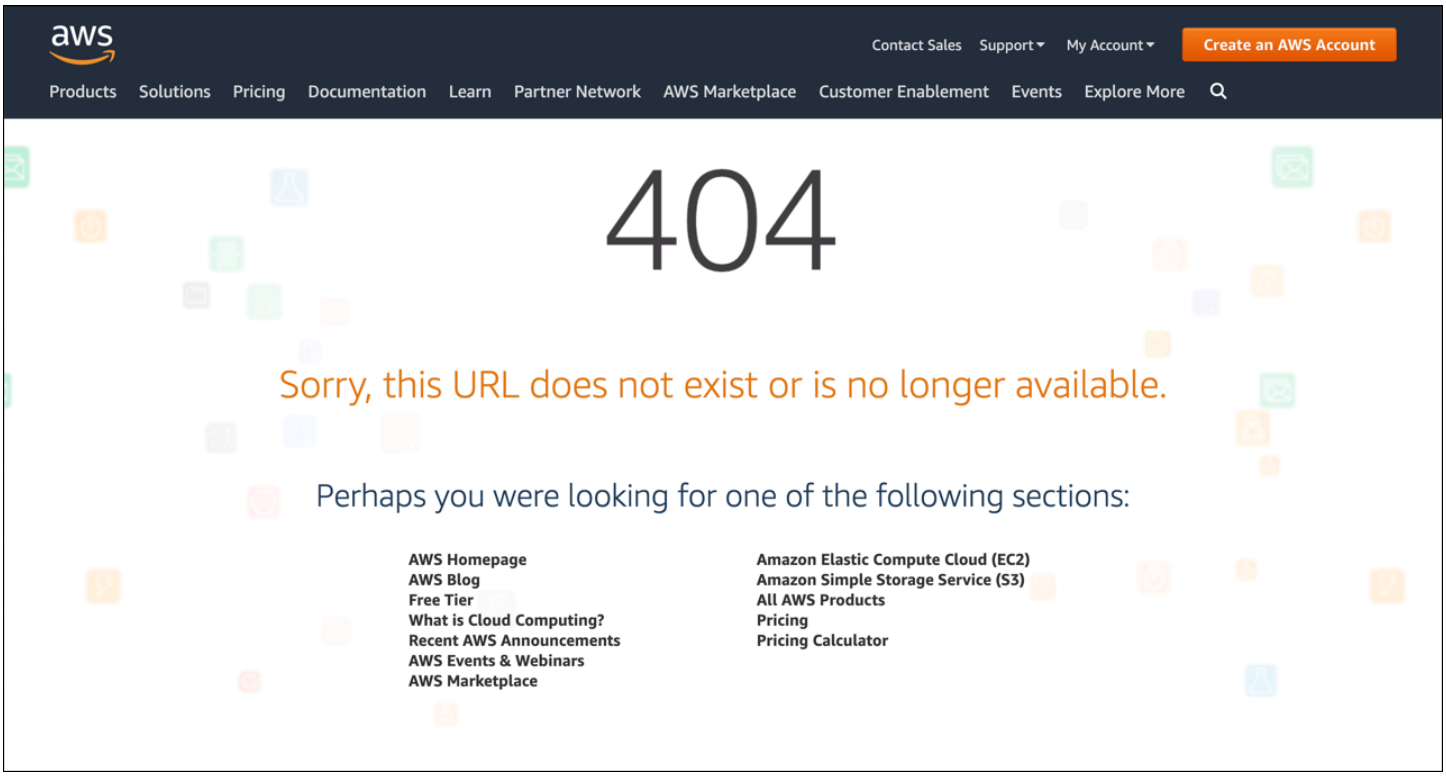
- AWS::CloudFront::Distribution CustomErrorResponse in the *AWS CloudFormation User Guide*
- CustomErrorResponse in the *Amazon CloudFront API Reference*

Create a custom error page for specific HTTP status codes

If you'd rather display a custom error message instead of the default message—for example, a page that uses the same formatting as the rest of your website—you can have CloudFront return to the viewer an object (such as an HTML file) that contains your custom error message.

To specify the file that you want to return and the errors for which the file should be returned, you update your CloudFront distribution to specify those values. For more information, see Configure error response behavior.

For example, the following is a custom error page:



You can specify a different object for each supported HTTP status code, or you can use the same object for all of the supported status codes. You can choose to specify custom error pages for some status codes and not for others.

The objects that you're serving through CloudFront can be unavailable for a variety of reasons. These fall into two broad categories:

- *Client errors* indicate a problem with the request. For example, an object with the specified name isn't available, or the user doesn't have the permissions required to get an object in your Amazon S3 bucket. When a client error occurs, the origin returns an HTTP status code in the 4xx range to CloudFront.
- *Server errors* indicate a problem with the origin server. For example, the HTTP server is busy or unavailable. When a server error occurs, either your origin server returns an HTTP status code in the 5xx range to CloudFront, or CloudFront doesn't get a response from your origin server for a certain period of time and assumes a 504 status code (Gateway Timeout).

The HTTP status codes for which CloudFront can return a custom error page include the following:

- 400, 403, 404, 405, 414, 416
- 500, 501, 502, 503, 504

Notes

- If CloudFront detects that the request might be unsafe, CloudFront returns a 400 (Bad Request) error instead of a custom error page.
- You can create a custom error page for HTTP status code 416 (Requested Range Not Satisfiable), and you can change the HTTP status code that CloudFront returns to viewers when your origin returns a status code 416 to CloudFront. For more information, see Change response codes returned by CloudFront. However, CloudFront doesn't cache status code 416 responses, so even if you specify a value for **Error Caching Minimum TTL** for status code 416, CloudFront doesn't use it.
- In some cases, CloudFront doesn't return a custom error page for the HTTP 503 status code even if you configure CloudFront to do so. If the CloudFront error code is `Capacity Exceeded` or `Limit Exceeded`, CloudFront returns a 503 status code to the viewer without using your custom error page.
- If you created a custom error page, CloudFront will return `Connection: close` or `Connection: keep-alive` for the following response codes:
  - CloudFront returns `Connection: close` for status codes: 400, 405, 414, 416, 500, 501
  - CloudFront returns `Connection: keep-alive` for status codes: 403, 404, 502, 503, 504

For a detailed explanation of how CloudFront handles error responses from your origin, see How CloudFront processes HTTP 4xx and 5xx status codes from your origin.

Store objects and custom error pages in different locations

If you want to store your objects and your custom error pages in different locations, your distribution must include a cache behavior for which the following is true:

- The value of **Path Pattern** matches the path to your custom error messages. For example, suppose you saved custom error pages for 4xx errors in an Amazon S3 bucket in a directory named `/4xx-errors`. Your distribution must include a cache behavior for which the path pattern routes requests for your custom error pages to that location, for example, `/4xx-errors/*`.
- The value of **Origin** specifies the value of **Origin ID** for the origin that contains your custom error pages.

For more information, see Cache behavior settings.

Change response codes returned by CloudFront

You can configure CloudFront to return a different HTTP status code to the viewer than what CloudFront received from the origin. For example, if your origin returns a 500 status code to CloudFront, you might want CloudFront to return a custom error page and a 200 status code (OK) to the viewer. There are a variety of reasons that you might want CloudFront to return a status code to the viewer that is different from the one that your origin returned to CloudFront:

- Some internet devices (some firewalls and corporate proxies, for example) intercept HTTP 4xx and 5xx status codes and prevent the response from being returned to the viewer. In this scenario, if you substitute `200`, the response is not intercepted.
- If you don't care about distinguishing among different client errors or server errors, you can specify `400` or `500` as the value that CloudFront returns for all 4xx or 5xx status codes.
- You might want to return a `200` status code (OK) and a static website so your customers don't know that your website is down.

If you enable CloudFront standard logs and you configure CloudFront to change the HTTP status code in the response, the value of the `sc-status` column in the logs contains the status code that you specify. However, the value of the `x-edge-result-type` column is not affected. It contains

the result type of the response from the origin. For example, suppose you configure CloudFront to return a status code of `200` to the viewer when the origin returns `404` (Not Found) to CloudFront. When the origin responds to a request with a `404` status code, the value in the `sc-status` column in the log will be `200`, but the value in the `x-edge-result-type` column will be `Error`.

You can configure CloudFront to return any of the following HTTP status codes along with a custom error page:

- 200
- 400, 403, 404, 405, 414, 416
- 500, 501, 502, 503, 504

Control how long CloudFront caches errors

CloudFront caches error responses for a default duration of 10 seconds. CloudFront then submits the next request for the object to your origin to see if the problem that caused the error has been resolved and the requested object is available.

You can specify the error-caching duration—the **Error Caching Minimum TTL**—for each 4xx and 5xx status code that CloudFront caches. (For more information, see HTTP 4xx and 5xx status codes that CloudFront caches.) When you specify a duration, note the following:

- If you specify a short error-caching duration, CloudFront forwards more requests to your origin than if you specify a longer duration. For 5xx errors, this might aggravate the problem that originally caused your origin to return an error.
- When your origin returns an error for an object, CloudFront responds to requests for the object either with the error response or with your custom error page until the error-caching duration elapses. If you specify a long error-caching duration, CloudFront might continue to respond to requests with an error response or your custom error page for a long time after the object becomes available again.

> **Note**
>
> You can create a custom error page for HTTP status code 416 (Requested Range Not Satisfiable), and you can change the HTTP status code that CloudFront returns to viewers when your origin returns a status code 416 to CloudFront. (For more information, see Change response codes returned by CloudFront.) However, CloudFront doesn't cache status

code 416 responses, so even if you specify a value for **Error Caching Minimum TTL** for status code 416, CloudFront doesn't use it.

If you want to control how long CloudFront caches errors for individual objects, you can configure your origin server to add the applicable header to the error response for that object.

If the origin adds a `Cache-Control: max-age` or `Cache-Control: s-maxage` directive, or an `Expires` header, CloudFront caches error responses for the greater of the value in the header or the **Error Caching Minimum TTL**.

Note

The `Cache-Control: max-age` and `Cache-Control: s-maxage` values cannot be greater than the **Maximum TTL** value set for the cache behavior for which the error page is being fetched.

If the origin adds a `Cache-Control: no-store`, `Cache-Control: no-cache`, or `Cache-Control: private` directive for error codes 404, 410, 414, or 501, CloudFront does not cache the error response. For all other error codes, CloudFront ignores the `no-store`, `no-cache`, and `private` directives and caches the error response for the value of **Error Caching Minimum TTL**.

If the origin adds other `Cache-Control` directives or adds no headers, CloudFront caches error responses for the value of **Error Caching Minimum TTL**.

If the expiration time for a 4xx or 5xx status code for an object is longer than you want, and the object is available again, you can invalidate cached error code by using the URL of the requested object. If your origin is returning an error response for multiple objects, you need to invalidate each object separately. For more information about invalidating objects, see Invalidate files to remove content.

If you have caching enabled for an S3 bucket origin, and you configure an error caching minimum TTL of 0 seconds in your CloudFront distribution, you will still see an error caching minimum TTL of 1 second for S3 origin errors. CloudFront does this to protect your origin from DDoS attacks. This doesn't apply to other types of origins.

Add, remove, or replace content that CloudFront distributes

This section explains how to make sure CloudFront can access the content that you want to be served to your viewers, how to specify the objects in your website or in your application, and how to remove or replace content.

Topics

- Add and access content that CloudFront distributes
- Use file versioning to update or remove content with a CloudFront distribution
- Customize the URL format for files in CloudFront
- Specify a default root object
- Invalidate files to remove content
- Serve compressed files

Add and access content that CloudFront distributes

When you want CloudFront to distribute content (objects), you add files to one of the origins that you specified for the distribution, and you expose a CloudFront link to the files. A CloudFront edge location doesn't fetch the new files from an origin until the edge location receives viewer requests for them. For more information, see How CloudFront delivers content.

When you add a file that you want CloudFront to distribute, make sure that you add it to one of the Amazon S3 buckets specified in your distribution or, for a custom origin, to a directory in the specified domain. In addition, confirm that the path pattern in the applicable cache behavior sends requests to the correct origin.

For example, suppose the path pattern for a cache behavior is `*.html`. If you don't have any other cache behaviors configured to forward requests to that origin, CloudFront will only forward `*.html` files. In this scenario, for example, CloudFront will never distribute .jpg files that you upload to the origin, because you haven't created a cache behavior that includes .jpg files.

CloudFront servers don't determine the MIME type for the objects that they serve. When you upload a file to your origin, we recommend that you set the `Content-Type` header field for it.

Use file versioning to update or remove content with a CloudFront distribution

To update existing content that CloudFront is set up to distribute for you, we recommend that you use a version identifier in file names or in folder names. This helps give you control over managing the content that CloudFront serves.

Update existing files using versioned file names

When you update existing files in a CloudFront distribution, we recommend that you include some sort of version identifier either in your file names or in your directory names to give yourself better control over your content. This identifier might be a date-time stamp, a sequential number, or some other method of distinguishing two versions of the same object.

For example, instead of naming a graphic file image.jpg, you might call it image_1.jpg. When you want to start serving a new version of the file, you'd name the new file image_2.jpg, and you'd update the links in your web application or website to point to image_2.jpg. Alternatively, you might put all graphics in an images_v1 directory and, when you want to start serving new versions of one or more graphics, you'd create a new images_v2 directory, and you'd update your links to point to that directory. With versioning, you don't have to wait for an object to expire before CloudFront begins to serve a new version of it, and you don't have to pay for object invalidation.

Even if you version your files, we still recommend that you set an expiration date. For more information, see Manage how long content stays in the cache (expiration).

> **Note**
>
> Specifying versioned file names or directory names is not related to Amazon S3 object versioning.

Remove content so CloudFront won’t distribute it

You can remove files from your origin that you no longer want to be included in your CloudFront distribution. However, CloudFront will continue to show viewers content from the edge cache until the files expire.

If you want to remove a file right away, you must do one of the following:

- **Use file versioning.** When you use versioning, different versions of a file have different names that you can use in your CloudFront distribution, to change which file is returned to viewers. For more information, see Update existing files using versioned file names.
- **Invalidate the file.** For more information, see Invalidate files to remove content .

Customize the URL format for files in CloudFront

After you set up your origin with the objects (content) that you want CloudFront to serve to your viewers, you must use the correct URLs to reference those objects in your website or application code so that CloudFront can serve it.

The domain name that you use in the URLs for objects on your web pages or in your web application can be either of the following:

- The domain name, such as `d111111abcdef8.cloudfront.net`, that CloudFront automatically assigns when you create a distribution
- Your own domain name, such as `example.com`

For example, you might use one of the following URLs to return the file `image.jpg`:

```
https://d111111abcdef8.cloudfront.net/images/image.jpg
```

```
https://example.com/images/image.jpg
```

You use the same URL format whether you store the content in Amazon S3 buckets or at a custom origin, like one of your own web servers.

> **Note**
>
> The URL format depends in part on the value that you specify for **Origin Path** in your distribution. This value gives CloudFront a top directory path for your objects. For more information about setting the origin path when you create a distribution, see Origin path.

For more information about URL formats, see the following sections.

Use your own domain name (example.com)

Instead of using the default domain name that CloudFront assigns for you when you create a distribution, you can add an alternate domain name that's easier to work with, like `example.com`. By setting up your own domain name with CloudFront, you can use a URL like this for objects in your distribution:

```
https://example.com/images/image.jpg
```

If you plan to use HTTPS between viewers and CloudFront, see Use alternate domain names and HTTPS.

Use a trailing slash (/) in URLs

When you specify URLs for directories in your CloudFront distribution, choose either to always use a trailing slash or to never use a trailing slash. For example, choose only one of the following formats for all of your URLs:

```
https://d111111abcdef8.cloudfront.net/images/
```

```
https://d111111abcdef8.cloudfront.net/images
```

Why does it matter?

Both formats work to link to CloudFront objects, but being consistent can help prevent issues when you want to invalidate a directory later. CloudFront stores URLs exactly as they are defined, including trailing slashes. So if your format is inconsistent, you'll need to invalidate directory URLs with and without the slash, to make sure that CloudFront removes the directory.

It's inconvenient to have to invalidate both URL formats, and it can lead to additional costs. That's because if you must double up invalidations to cover both types of URLs, you might exceed the maximum number of free invalidations allowed for the month. And if that happens, you'll have to pay for all the invalidations, even if only one format for each directory URL exists in CloudFront.

Create signed URLs for restricted content

If you have content that you want to restrict access to, you can create signed URLs. For example, if you want to distribute your content only to users who have authenticated, you can create URLs that are valid only for a specified time period or that are available only from a specified IP address. For more information, see Serve private content with signed URLs and signed cookies.

Specify a default root object

You can configure CloudFront to return a specific object (the default root object) when a user (viewer) requests the root URL for your distribution instead of requesting an object in your distribution. You can use a default root object to avoid exposing the contents of your distribution.

Contents

- How to specify a default root object
- How default root object works
- How CloudFront works if you don't define a root object

How to specify a default root object

To avoid exposing the contents of your distribution or returning an error, specify a default root object for your distribution. You can specify the exact file name or the path to the file. For example, if your root object is an `index.html` file, you can specify that file name. If your `index.html` file is in another folder, specify the path instead such as *`exampleFolderName`*`/index.html`. If you set a path to the default root object, viewer requests to the root URL of the distribution will return the specified file from that path. You can use a file path to have more flexibility to organize your content at the origin, as your default root object can be in a folder instead of at the root level.

To specify a default root object for your distribution

1. Upload the default root object to the origin that your distribution points to.

   The file can be any type supported by CloudFront. For a list of constraints on the file name, see the `DefaultRootObject` element in DistributionConfig of the *Amazon CloudFront API Reference*.

   > **Note**
   >
   > If the file name of the default root object is too long or contains an invalid character, CloudFront returns the error `HTTP 400 Bad Request - InvalidDefaultRootObject`. In addition, CloudFront caches the code for 10 seconds (by default) and writes the results to the access logs.

2. Confirm that the permissions for the object grant CloudFront at least read access.

For more information about Amazon S3 permissions, see Identity and access management in Amazon S3 in the *Amazon Simple Storage Service User Guide* .

3. Update your distribution to refer to the default root object by using the CloudFront console or the CloudFront API.

   To specify a default root object by using the CloudFront console:

   a. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

   b. In the list of distributions in the top pane, select the distribution to update.

   c. In the **Settings** pane, on the **General** tab, choose **Edit**.

   d. In the **Edit settings** dialog box, in the **Default root object** field, enter the file name or path to the default root object.

   > **Tip**
   >
   > Your string can't begin with a forward slash (/). Only specify the object name or the path to the object. For example, use `index.html` or *`exampleFolderName`*`/index.html`. Specifying a **`/`** *`exampleFolderName`*`/index.html` or **`/`**`index.html` can lead to a 403 Access Denied error.

   e. Choose **Save changes**.

   To update your configuration using the CloudFront API, specify a value for the `DefaultRootObject` element in your distribution. For information about using the CloudFront API to specify a default root object, see UpdateDistribution in the *Amazon CloudFront API Reference*.

4. Confirm that you have enabled the default root object by requesting your root URL. If your browser doesn't display the default root object, perform the following steps:

   a. Confirm that your distribution is fully deployed by viewing the status of your distribution in the CloudFront console.

   b. Repeat steps 2 and 3 to verify that you granted the correct permissions and that you correctly updated the configuration of your distribution to specify the default root object.

How default root object works

Suppose the following request points to the object `image.jpg`:

`https://d111111abcdef8.cloudfront.net/image.jpg`

In contrast, the following request points to the root URL of the same distribution instead of to a specific object, as in the first example:

`https://d111111abcdef8.cloudfront.net/`

When you define a default root object, an end-user request that calls the root of your distribution returns the default root object. For example, if you designate the file `index.html` as your default root object, a request for:

`https://d111111abcdef8.cloudfront.net/`

Returns:

`https://d111111abcdef8.cloudfront.net/index.html`

> **Note**
>
> CloudFront does not determine whether a URL with multiple trailing slashes (`https://d111111abcdef8.cloudfront.net///`) is equivalent to `https://d111111abcdef8.cloudfront.net/`. Your origin server makes that comparison.

If you define a default root object, an end-user request for a subdirectory of your distribution does not return the default root object. For example, suppose `index.html` is your default root object and that CloudFront receives an end-user request for the `install` directory under your CloudFront distribution:

`https://d111111abcdef8.cloudfront.net/install/`

CloudFront does not return the default root object even if a copy of `index.html` appears in the `install` directory. However, if you specified a *path* to your default root object, (`install/index.html`)CloudFront will return the default root object for end-user requests for the `install` directory

If you configure your distribution to allow all of the HTTP methods that CloudFront supports, the default root object applies to all methods. For example, if your default root object is index.php and you write your application to submit a POST request to the root of your domain (https://example.com), CloudFront sends the request to https://example.com/index.php.

The behavior of CloudFront default root objects is different from the behavior of Amazon S3 index documents. When you configure an Amazon S3 bucket as a website and specify the index document, Amazon S3 returns the index document even if a user requests a subdirectory in the bucket. (A copy of the index document must appear in every subdirectory.) For more information about configuring Amazon S3 buckets as websites and about index documents, see the Hosting Websites on Amazon S3 chapter in the *Amazon Simple Storage Service User Guide*.

> **Important**
>
> Remember that a default root object applies only to your CloudFront distribution. You still need to manage security for your origin. For example, if you are using an Amazon S3 origin, you still need to set your Amazon S3 bucket ACLs appropriately to make sure the level of access you want on your bucket.

How CloudFront works if you don't define a root object

If you don't define a default root object, requests for the root of your distribution pass to your origin server. If you are using an Amazon S3 origin, any of the following might be returned:

- **A list of the contents of your Amazon S3 bucket** – Under any of the following conditions, the contents of your origin are visible to anyone who uses CloudFront to access your distribution:
  - Your bucket is not properly configured.
  - The Amazon S3 permissions on the bucket associated with your distribution and on the objects in the bucket grant access to *everyone*.
  - An end user accesses your origin using your origin root URL.
- **A list of the private contents of your origin** – If you configure your origin as a private distribution (only you and CloudFront have access), the contents of the Amazon S3 bucket associated with your distribution are visible to anyone who has the credentials to access your distribution through CloudFront. In this case, users are not able to access your content through your origin root URL. For more information about distributing private content, see the section called "Restrict content with signed URLs and signed cookies".

- `Error 403 Forbidden`—CloudFront returns this error if the permissions on the Amazon S3 bucket associated with your distribution or the permissions on the objects in that bucket deny access to CloudFront and to everyone.

Invalidate files to remove content

If you need to remove a file from CloudFront edge caches before it expires, you can do one of the following:

- Invalidate the file from edge caches. The next time a viewer requests the file, CloudFront returns to the origin to fetch the latest version of the file.
- Use file versioning to serve a different version of the file that has a different name. For more information, see Update existing files using versioned file names.

Topics

- Choose between invalidating files and using versioned file names
- Determine which paths to invalidate
- What you need to know when invalidating paths
- Invalidating content by cache tags
- What you need to know when invalidating tags
- Invalidate files
- Pay for file invalidation

Choose between invalidating files and using versioned file names

To control the versions of files that are served from your distribution, you can either invalidate files or give them versioned file names. If you want to update your files frequently, we recommend that you primarily use file versioning for the following reasons:

- Versioning enables you to control which file a request returns even when the user has a version cached either locally or behind a corporate caching proxy. If you invalidate the file, the user might continue to see the old version until it expires from those caches.
- CloudFront access logs include the names of your files, so versioning makes it easier to analyze the results of file changes.

- Versioning provides a way to serve different versions of files to different users.
- Versioning simplifies rolling forward and back between file revisions.
- Versioning is less expensive. You still have to pay for CloudFront to transfer new versions of your files to edge locations, but you don't have to pay for invalidating files.

For more information about file versioning, see Update existing files using versioned file names.

Determine which paths to invalidate

If you want to invalidate multiple files such as all of the files in a directory or all files that begin with the same characters, you can include the * wildcard at the end of the invalidation path. For more information about using the * wildcard, see Invalidation paths.

To invalidate files, you can specify either the path for individual files or a path that ends with the * wildcard, which might apply to one file or to many, as shown in the following examples:

- `/images/image1.jpg`
- `/images/image*`
- `/images/*`

If you want to invalidate selected files but your users don't necessarily access every file on your origin, you can determine which files viewers have requested from CloudFront and invalidate only those files. To determine which files viewers have requested, enable CloudFront access logging. For more information about access logs, see Access logs (standard logs).

What you need to know when invalidating paths

When you specify a file to invalidate, refer to the following information:

Case sensitivity

Invalidation paths are case sensitive. For example, `/images/image.jpg` and `/images/Image.jpg` specify two different files.

Changing the URI by using a Lambda function

If your CloudFront distribution triggers a Lambda function on viewer request events, and if the function changes the URI of the requested file, we recommend that you invalidate both URIs to remove the file from CloudFront edge caches:

- The URI in the viewer request
- The URI after the function changed it

Example

Suppose your Lambda function changes the URI for a file from:

`https://d111111abcdef8.cloudfront.net/index.html`

To a URI that includes a language directory:

`https://d111111abcdef8.cloudfront.net/en/index.html`

To invalidate the file, you must specify the following paths:

- `/index.html`
- `/en/index.html`

For more information, see Invalidation paths.

Default root object

To invalidate the default root object (file), specify the path the same way that you specify the path for any other file. For more information, see How default root object works.

Forwarding cookies

If you configured CloudFront to forward cookies to your origin, CloudFront edge caches might contain several versions of the file. When you invalidate a file, CloudFront invalidates every cached version of the file regardless of its associated cookies. You can't selectively invalidate some versions and not others based on the associated cookies. For more information, see Cache content based on cookies.

Forwarding headers

If you configured CloudFront to forward a list of headers to your origin and to cache based on the values of the headers, CloudFront edge caches might contain several versions of the file. When you invalidate a file, CloudFront invalidates every cached version of the file regardless of the header values. You can't selectively invalidate some versions and not others based on header values. (If you configure CloudFront to forward all headers to your origin, CloudFront doesn't cache your files.) For more information, see Cache content based on request headers.

Forwarding query strings

If you configured CloudFront to forward query strings to your origin, you must include the query strings when invalidating files, as shown in the following examples:

- `/images/image.jpg?parameter1=a`
- `/images/image.jpg?parameter1=b`

If client requests include five different query strings for the same file, you can either invalidate the file five times, once for each query string, or you can use the * wildcard in the invalidation path, as shown in the following example:

```
/images/image.jpg*
```

For more information about using wildcards in the invalidation path, see Invalidation paths.

For more information about query strings, see Cache content based on query string parameters.

To determine which query strings are in use, you can enable CloudFront logging. For more information, see Access logs (standard logs).

Maximum allowed

For more information about the maximum number of invalidations allowed, see Quotas on invalidations.

Microsoft Smooth Streaming files

You can't invalidate media files in the Microsoft Smooth Streaming format when you have enabled Smooth Streaming for the corresponding cache behavior.

Non-ASCII or unsafe characters in the path

If the path includes non-ASCII characters or unsafe characters as defined in RFC 1738, URL-encode those characters. Don't URL-encode any other characters in the path, or CloudFront won't invalidate the old version of the updated file.

> **Important**
>
> Don't use the ~ character in your path. CloudFront doesn't support this character for invalidations, whether it's URL-encoded or not.

Invalidation paths

The path is relative to the distribution. For example, to invalidate the file at `https://d111111abcdef8.cloudfront.net/images/image2.jpg`, you would specify `/images/image2.jpg`.

> **Note**
>
> In the CloudFront console, you can omit the leading slash in the path, like this: `images/image2.jpg`. When you use the CloudFront API directly, invalidation paths must begin with a leading slash.

You can also invalidate multiple files simultaneously by using the * wildcard. The *, which replaces 0 or more characters, must be the last character in the invalidation path.

> **Important**
>
> To use wildcards (*) in the invalidation, you must put the wildcard at the end of the path. Asterisks (*) inserted anywhere else are treated as a literal character match instead of a wildcard invalidation.

If you use the AWS Command Line Interface (AWS CLI) to invalidate files and you specify a path that includes the * wildcard, you must use quotes (") around the path like `"/*"`.

The maximum length of a path is 4,000 characters.

Example: Invalidation paths

- To invalidate all files in a directory, all of its subdirectories, and all files in the directory and subdirectories:

  /*directory-path* /*

- To invalidate a directory, all of its subdirectories, and all files in the directory and subdirectories:

  /*directory-path* *

- To invalidate all files that have the same name but different file name extensions, such as logo.jpg, logo.png, and logo.gif:

/directory-path/file-name.*

- To invalidate all of the files in a directory for which the file name starts with the same characters (such as all of the files for a video in HLS format), regardless of the file name extension:

  */directory-path/initial-characters-in-file-name**

- When you configure CloudFront to cache based on query string parameters and you want to invalidate every version of a file:

  */directory-path/file-name.file-name-extension**

- To invalidate all files in a distribution:

  /*

For information about invalidating files if you use a Lambda function to change the URI, see Changing the URI Using a Lambda Function.

If the invalidation path is a directory and if you have not standardized on a method for specifying directories—with or without a trailing slash (/)—we recommend that you invalidate the directory both with and without a trailing slash, for example, `/images` and `/images/`.

Signed URLs

If you are using signed URLs, invalidate a file by including only the portion of the URL before the question mark (?).

Invalidating content by cache tags

Amazon CloudFront supports tag invalidations, which lets you invalidate cached objects based on semantic tags rather than URL paths. This gives you flexible control over cache invalidation without requiring your URL structure to match your invalidation strategy.

Before you invalidate content by cache tags, review the requirements and considerations in What you need to know when invalidating tags.

How tag invalidation works

1. **Configure your distribution**: Add a `CacheTagConfig` to your distribution specifying the HTTP header name that your origin uses to return cache tags.

2. **Tag objects at your origin**: When returning an object you want to cache with tags, configure your origin to include the specified header in HTTP responses with comma-separated tag values.

```
HTTP/1.1 200 OK
Content-Type: text/html
x-amz-meta-cache-tag: product:electronics, category:tv, brand:example
Cache-Control: max-age=3600
```

> **Note**
>
> For S3 origins, you can attach cache tags to your S3 objects as metadata. You can add a metadata entry with a key of your choice (example: `cache-tag`) and a comma-separated list of tags as the value (example: `product:electronics, category:tv, brand:example`).
> S3 surfaces object metadata as response headers prefixed with `x-amz-meta-<Key>`, so a metadata key of `cache-tag` would be returned as an `x-amz-meta-cache-tag` header. You can set the `HeaderName` in your `CacheTagConfig` to `x-amz-meta-cache-tag` to enable sending invalidations to these tags.

> **Note**
>
> Alternatively, you can attach an origin response Lambda@Edge function to your cache behavior to add cache tag headers. If using Lambda@Edge, the header name does not need to follow the `x-amz-meta-<Key>` format.

3. **Invalidate by tag**: Use the `CreateInvalidation` API with the # prefix to invalidate all objects cached with the specified tag.

```
aws cloudfront create-invalidation \
  --distribution-id distribution_ID \
  --paths "#product:electronics"
```

This invalidates ALL cached objects that contain the tag `product:electronics`, regardless of their URL path.

Tag format requirements

Origin response header value:

- Tags are comma-separated in the header value.
- Leading and trailing spaces around each tag are trimmed.
- Both `tag1,tag2,tag3` and `tag1, tag2, tag3` are valid and equivalent.

Individual tag values:

- ASCII visible characters (33–126)
- No control characters, spaces, or commas
- Case-insensitive
- Maximum 256 characters per tag
- Maximum 50 tags per object (additional tags are ignored)

Configuring a distribution for tag invalidations

To enable tag invalidations, add a `CacheTagConfig` to your distribution configuration:

```
aws cloudfront create-distribution \
  --distribution-config '{
    "CallerReference": "my-distribution",
    "CacheTagConfig": {
      "HeaderName": "x-amz-meta-cache-tag"
    },
    "DefaultCacheBehavior": {
      "TargetOriginId": "myOrigin",
      "ViewerProtocolPolicy": "redirect-to-https",
      "CachePolicyId": "658327ea-f89d-4fab-a63d-7e88639e58f6"
    },
    "Origins": { ... },
    "Enabled": true,
    "Comment": "Distribution with tag support"
  }'
```

Creating tag invalidations

Use the # prefix in the `--paths` parameter to specify tags:

```
# Invalidate all objects tagged with "user1"
aws cloudfront create-invalidation \
  --distribution-id distribution_ID \
  --paths "#user1"

# Invalidate objects matching any of multiple tags (OR logic)
aws cloudfront create-invalidation \
  --distribution-id distribution_ID \
  --paths "#user1" "#product-category:electronics"

# Mix path and tag invalidations in one batch
aws cloudfront create-invalidation \
  --distribution-id distribution_ID \
  --paths "/index.html" "#user1" "/images/*" "#product-category:electronics"
```

Checking invalidation status

Use `GetInvalidation` to check the status. The response includes both path and tag items:

```
aws cloudfront get-invalidation \
  --distribution-id distribution_ID \
  --id invalidation_ID
```

Important considerations

- **Opt-in required**: Tag invalidation only works on distributions that have `CacheTagConfig` configured. Distributions without this configuration ignore cache tag headers from the origin.
- **Tag processing limits**: CloudFront processes up to 50 tags per cached object. If an origin response contains more than 50 tags, the additional tags beyond the limit are not stored.
- **Changing the header name**: Tag invalidations are looked up against the current `CacheTagConfig` configuration. If you change the `HeaderName` in the `CacheTagConfig`, invalidations issued to objects cached with tags under the old header name will no longer be evaluated. If you need to change the header name, start returning both the new and the old cache tags headers with your objects, then issue a path invalidation (for example, `/*`) or invalidate existing tags before changing the header name to avoid serving stale content. Once that's done you can stop sending the old cache tags header with your objects.
- **Removing CacheTagConfig**: When you remove `CacheTagConfig` from a distribution, CloudFront stops extracting tags from origin responses. Existing cached objects with tags are served normally until they expire or are invalidated by path.

- **Backward compatibility**: Existing path and wildcard invalidations continue to work unchanged. Tag invalidations are additive — you can use both methods on the same distribution.
- **Distribution tenants**: Tag invalidation is also supported for distribution tenants by using the `CreateInvalidationForDistributionTenant` API.

Use case examples

Example 1: E-commerce product catalog

An e-commerce site tags cached product pages with product, category, and brand:

```
x-amz-meta-cache-tag: category:electronics, brand:acme, product:12345
```

When ACME updates their branding, invalidate all ACME products at once:

```
aws cloudfront create-invalidation \
  --distribution-id distribution_ID \
  --paths "#brand:acme"
```

Example 2: User-generated content platform

A platform tags cached content with the owner's user ID:

```
x-amz-meta-cache-tag: user:12345, content-type:image
```

When a user closes their account, invalidate all their content after you've removed it from the origin:

```
aws cloudfront create-invalidation \
  --distribution-id distribution_ID \
  --paths "#user:12345"
```

Example 3: Content versioning

A CMS tags content with version identifiers:

```
x-amz-meta-cache-tag: version:v2, template:homepage
```

When deploying a new version, invalidate all v2 content:

```
aws cloudfront create-invalidation \
  --distribution-id distribution_ID \
  --paths "#version:v2"
```

What you need to know when invalidating tags

When you specify a tag to invalidate, refer to the following information:

Opt-in required

Tag invalidation only works on distributions that have `CacheTagConfig` configured. Distributions without this configuration ignore cache tag headers from the origin. For more information, see Invalidating content by cache tags.

Tag invalidation prefix

Tag invalidation items must begin with the # character. For example, to invalidate all cached objects tagged with `product:electronics`, specify `#product:electronics` in the `Paths.Items` list.

Case sensitivity

Tag values are case-insensitive. For example, `#Product:Electronics` and `#product:electronics` refer to the same tag.

Tag format

Each tag value must contain only ASCII visible characters (33–126), excluding commas. The maximum length of a tag is 256 characters.

No wildcard support

Unlike path invalidations, tag invalidation items do not support the * wildcard. Each #-prefixed item matches the exact tag value.

Maximum tags per cached object

CloudFront processes up to 50 tags per cached object. If an origin response contains more than 50 tags in the configured header, the additional tags beyond the limit are not stored and cannot be used for invalidation.

Forwarding cookies, headers, and query strings

When you invalidate by tag, CloudFront invalidates every cached variant of every object that carries the specified tag, regardless of associated cookies, headers, or query string parameters. You can't selectively invalidate some variants and not others.

Mixed path and tag invalidations

You can include both path items (for example, `/images/*`) and tag items (for example, `#brand:acme`) in the same invalidation batch in a `CreateInvalidation` request. Each item counts as one invalidation path toward your limits.

Maximum allowed

Tag invalidation items count toward the same concurrent invalidation limits as path items. Each tag item counts as one invalidation path. For more information, see Quotas on invalidations.

Changing the header name

If you change the `HeaderName` in `CacheTagConfig`, invalidations issued against tags extracted under the old header name will no longer be evaluated. Before changing the header name:

1. Start returning both the old and new cache tag headers from your origin.
2. Issue a path invalidation (for example, `/*`) or invalidate existing tags.
3. Update `CacheTagConfig` with the new `HeaderName` .
4. Stop returning the old header from your origin.

Distribution tenants

Tag invalidation is also supported for distribution tenants by using the `CreateInvalidationForDistributionTenant` API.

Invalidate files

You can use the CloudFront console to create and run an invalidation, display a list of the invalidations that you submitted previously, and display detailed information about an individual invalidation. You can also copy an existing invalidation, edit the list of file paths, and run the edited invalidation. You can't remove invalidations from the list.

Contents

- Invalidate files

- Copy, edit, and rerun an existing invalidation
- Cancel invalidations
- List invalidations
- Display information about an invalidation

Invalidate files

To invalidate files using the CloudFront console, do the following.

Console

To invalidate files (console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the distribution for which you want to invalidate files.
3. Choose the **Invalidations** tab.
4. Choose **Create invalidation**.
5. For the files that you want to invalidate, enter one invalidation path per line. For information about specifying invalidation paths, see What you need to know when invalidating paths.

   > **Important**
   >
   > Specify file paths carefully. You can't cancel an invalidation request after you start it.

6. Choose **Create invalidation**.

CloudFront API

To learn about invalidating objects and displaying information about invalidations, see the following topics in the *Amazon CloudFront API Reference*:

- CreateInvalidation
- ListInvalidations

- GetInvalidation

> **Note**
>
> If you use the AWS Command Line Interface (AWS CLI) to invalidate files and you specify a path that includes the * wildcard, you must use quotes (") around the path, such as the following example:
>
> ```
> aws cloudfront create-invalidation --distribution-id distribution_ID --paths
>  "/*"
> ```

Copy, edit, and rerun an existing invalidation

You can copy an invalidation that you created previously, update the list of invalidation paths, and run the updated invalidation. You can't copy an existing invalidation, update the invalidation paths, and then save the updated invalidation without running it.

> **Important**
>
> If you copy an invalidation that is still in progress, update the list of invalidation paths, and then run the updated invalidation, CloudFront won't stop or delete the invalidation that you copied. If any invalidation paths appear in the original and in the copy, CloudFront will try to invalidate the files twice, and both invalidations will count against your maximum number of free invalidations for the month. If you already reached the maximum number of free invalidations, you will be charged for both invalidations of each file. For more information, see Quotas on invalidations.

To copy, edit, and rerun an existing invalidation

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Select the distribution that contains the invalidation that you want to copy.
3. Choose the **Invalidations** tab.
4. Choose the invalidation that you want to copy.

If you aren't sure which invalidation you want to copy, you can choose an invalidation and choose **View details** to display detailed information about that invalidation.

5. Choose **Copy to new**.
6. Update the list of invalidation paths if applicable.
7. Choose **Create invalidation**.

Cancel invalidations

When you submit an invalidation request to CloudFront, CloudFront forwards the request to all edge locations within a few seconds, and each edge location starts processing the invalidation immediately. As a result, you can't cancel an invalidation after you submit it.

List invalidations

You can display a list of the last 100 invalidations that you've created and run for a distribution by using the CloudFront console. If you want to get a list of more than 100 invalidations, use the `ListInvalidations` API operation. For more information, see ListInvalidations in the *Amazon CloudFront API Reference*.

To list invalidations

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Select the distribution for which you want to display a list of invalidations.
3. Choose the **Invalidations** tab.

> **Note**
>
> You can't remove invalidations from the list.

Display information about an invalidation

You can display detailed information about an invalidation, including distribution ID, invalidation ID, the status of the invalidation, the date and time that the invalidation was created, and a complete list of the invalidation paths.

To display information about an invalidation

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Select the distribution that contains the invalidation that you want to display detailed information for.
3. Choose the **Invalidations** tab.
4. Choose the applicable invalidation ID or select the invalidation ID and then choose **View details**.

Pay for file invalidation

The first 1,000 invalidation paths that you submit per month are free; you pay for each invalidation path over 1,000 in a month. An invalidation path can be for a single file (such as `/images/logo.jpg`) or for multiple files (such as `/images/*`). A path that includes the `*` wildcard counts as one path even if it causes CloudFront to invalidate thousands of files.

The maximum of 1,000 free invalidation paths per month applies to the total number of invalidation paths across all of the distributions that you create with one AWS account. For example, suppose you use the AWS account `john@example.com` to create three distributions, and you submit 600 invalidation paths for *each distribution* in a given month (for a total of 1,800 invalidation paths). In this case, AWS will charge you for the difference between the total invalidation paths and the 1000 free limit. In this example, AWS would charge you for 800 invalidation paths in that month.

The charge to submit an invalidation path is the same regardless of the number of files you're invalidating: a single file (`/images/logo.jpg`) or all of the files that are associated with a distribution (`/*`). Because you're charged per path in your invalidation request, even if you bundle multiple paths into a single request, each path is still counted individually for billing purposes.

For more information about invalidation pricing, see Amazon CloudFront Pricing. For more information about invalidation paths, see Invalidation paths.

Tag invalidation paths count toward the same 1,000 free invalidation paths per month allowance. For example, if you submit 500 path invalidations and 600 tag invalidations in a month (for a total of 1,100 invalidation paths), you are charged for 100 invalidation paths. A single tag invalidation item (for example, `#product:electronics`) counts as one path regardless of how many cached

objects it invalidates — the same as a wildcard path invalidation (for example, `/*`) counts as one path.

For more information about tag invalidation, see Invalidating content by cache tags.

Serve compressed files

When requested objects are compressed, downloads can be faster because the objects are smaller —in some cases, less than a quarter the size of the original. Faster downloads can result in faster rendering of webpages for your viewers, especially for JavaScript and CSS files. In addition, the cost of CloudFront data transfer is based on the total amount of data served. Serving compressed objects can be less expensive than serving them uncompressed.

Topics

- Configure CloudFront to compress objects
- How CloudFront compression works
- Conditions for compression
- File types that CloudFront compresses
- ETag header conversion

Configure CloudFront to compress objects

To configure CloudFront to compress objects, update the cache behavior that you want to serve the compressed objects.

To configure CloudFront to compress objects (console)

1. Sign in to the CloudFront console.
2. Choose your distribution and then choose the **Behavior** to edit.
3. For the **Compress objects automatically** setting, choose **Yes**.
4. Use a cache policy to specify the caching settings, and enable both **Gzip** and **Brotli** compression formats.

Notes

- You must use cache policies to use Brotli compression. Brotli doesn't support legacy cache settings.
- To enable compression by using CloudFormation or the CloudFront API, set the `Compress`, `EnableAcceptEncodingGzip`, and `EnableAcceptEncodingBrotli` parameters to `true`.

To understand how CloudFront compresses objects, see the following section.

How CloudFront compression works

1. A viewer requests an object. The viewer includes the `Accept-Encoding` HTTP header in the request, and the header value includes `gzip`, `br`, or both. This indicates that the viewer supports compressed objects. When the viewer supports both Gzip and Brotli, CloudFront uses Brotli.

   Note

   Chrome and Firefox web browsers support Brotli compression only when the request is sent using HTTPS. They don't support Brotli with HTTP requests.

2. At the edge location, CloudFront checks the cache for a compressed copy of the requested object.
3. Depending on whether the compressed object is in the cache or not, CloudFront does one of the following:
   - If the compressed object is already in the cache, CloudFront sends the object to the viewer and skips the remaining steps.
   - If the compressed object isn't in the cache, CloudFront forwards the request to the origin.

   Note

   If an uncompressed copy of the object is already in the cache, CloudFront might send it to the viewer without forwarding the request to the origin. For example, this can happen when CloudFront previously skipped compression. When this happens,

CloudFront caches the uncompressed object and continues to serve it until the object expires, is evicted, or is invalidated.

4. If the origin returns a compressed object (as indicated by the `Content-Encoding` header in the HTTP response), CloudFront sends the compressed object to the viewer, adds it to the cache, and skips the remaining steps. CloudFront doesn't compress the object again.
5. If the origin returns an uncompressed object to CloudFront without the `Content-Encoding` header in the HTTP response, CloudFront then determines whether the object can be compressed. For more information, see Conditions for compression.
6. If the object can be compressed, CloudFront compresses it, sends it to the viewer, and then adds it to the cache.
7. If there are subsequent viewer requests for the same object, CloudFront returns the first cached version. For example, if a viewer requests a specific cached object that uses Gzip compression, and the viewer *accepts* the Gzip format, subsequent requests to the same object will always return the Gzip version, even if the viewer accepts both Brotli and Gzip.

Some custom origins can also compress objects. Your origin might be able to compress objects that CloudFront doesn't compress. For more information, see File types that CloudFront compresses.

Conditions for compression

The following list provides more information about scenarios in which CloudFront doesn't compress objects.

Request uses HTTP 1.0

If a request to CloudFront uses HTTP 1.0, CloudFront removes the `Accept-Encoding` header and doesn't compress the object in the response.

`Accept-Encoding` request header

If the `Accept-Encoding` header is missing from the viewer request, or if it doesn't contain `gzip` or `br` as a value, CloudFront doesn't compress the object in the response. If the `Accept-Encoding` header includes additional values such as `deflate`, CloudFront removes them before forwarding the request to the origin.

When CloudFront is configured to compress objects, it includes the `Accept-Encoding` header in the cache key and in origin requests automatically.

Content is already cached when you configure CloudFront to compress objects

CloudFront compresses objects when it gets them from the origin. When you configure CloudFront to compress objects, CloudFront doesn't compress objects that are already cached in edge locations. In addition, when a cached object expires in an edge location and CloudFront forwards another request for the object to your origin, CloudFront doesn't compress the object when your origin returns an HTTP status code 304. This means that the edge location already has the latest version of the object. If you want CloudFront to compress objects that are already cached in edge locations, you need to invalidate those objects. For more information, see Invalidate files to remove content.

Origin is already configured to compress objects

If you configure CloudFront to compress objects and the origin also compresses objects, the origin should include a `Content-Encoding` header. This header indicates to CloudFront that the object is already compressed. When a response from an origin includes the `Content-Encoding` header, CloudFront doesn't compress the object, regardless of the header's value. CloudFront sends the response to the viewer and caches the object in the edge location.

File types that CloudFront compresses

For a complete list, see File types that CloudFront compresses.

Size of objects that CloudFront compresses

CloudFront compresses objects that are between 1,000 bytes and 10,000,000 bytes in size.

`Content-Length` header

The origin must include a `Content-Length` header in the response, which CloudFront uses to determine whether the size of the object is in the range that CloudFront compresses. If the `Content-Length` header is missing, contains an invalid value, or contains a value outside the range of sizes that CloudFront compresses, CloudFront doesn't compress the object. For more information about how CloudFront processes large objects that can exceed the size range, see How CloudFront processes partial requests for an object (range GETs).

HTTP status code of the response

CloudFront compresses objects only when the HTTP status code of the response is `200`, `403`, or `404`.

Response has no body

When the HTTP response from the origin has no body, there is nothing for CloudFront to compress.

ETag header

CloudFront sometimes modifies the ETag header in the HTTP response when it compresses objects. For more information, see the section called “ETag header conversion”.

CloudFront skips compression

CloudFront compresses objects on a best-effort basis. In rare cases, CloudFront skips compressing an object when CloudFront experiences high traffic load. CloudFront makes this decision based on a variety of factors, including host capacity. If CloudFront skips compression for an object, it caches the uncompressed object and continues to serve it to viewers until the object expires, is evicted, or is invalidated.

File types that CloudFront compresses

If you configure CloudFront to compress objects, CloudFront only compresses objects that have one of the following values in the `Content-Type` response header:

- `application/dash+xml`
- `application/eot`
- `application/font`
- `application/font-sfnt`
- `application/javascript`
- `application/json`
- `application/opentype`
- `application/otf`
- `application/pdf`
- `application/pkcs7-mime`
- `application/protobuf`
- `application/rss+xml`
- `application/truetype`

- application/ttf
- application/vnd.apple.mpegurl
- application/vnd.mapbox-vector-tile
- application/vnd.ms-fontobject
- application/wasm
- application/xhtml+xml
- application/xml
- application/x-font-opentype
- application/x-font-truetype
- application/x-font-ttf
- application/x-httpd-cgi
- application/x-javascript
- application/x-mpegurl
- application/x-opentype
- application/x-otf
- application/x-perl
- application/x-ttf
- font/eot
- font/opentype
- font/otf
- font/ttf
- image/svg+xml
- text/css
- text/csv
- text/html
- text/javascript
- text/js
- text/plain
- text/richtext
- text/tab-separated-values

- `text/xml`
- `text/x-component`
- `text/x-java-source`
- `text/x-script`
- `vnd.apple.mpegurl`

ETag header conversion

When the uncompressed object from the origin includes a valid, strong `ETag` HTTP header, and CloudFront compresses the object, CloudFront also converts the strong `ETag` header value to a weak `ETag`, and returns the weak `ETag` value to the viewer. Viewers can store the weak `ETag` value and use it to send conditional requests with the `If-None-Match` HTTP header. This allows viewers, CloudFront, and the origin to treat the compressed and uncompressed versions of an object as semantically equivalent, which reduces unnecessary data transfer.

A valid, strong `ETag` header value begins and ends with a double quote character ("). To convert the strong `ETag` value to a weak one, CloudFront adds the characters `W/` to the beginning of the strong `ETag` value.

When the object from the origin includes a weak `ETag` header value (a value that begins with the characters `W/`), CloudFront does not modify this value, and returns it to the viewer as received from the origin.

When the object from the origin includes an invalid `ETag` header value (the value does not begin with " or with `W/`), CloudFront removes the `ETag` header and returns the object to the viewer without the `ETag` response header.

For more information, see the following pages in the MDN web docs:

- Directives (`ETag` HTTP header)
- Weak validation (HTTP conditional requests)
- If-None-Match HTTP header

Use AWS WAF protections

You can use AWS WAF to protect your CloudFront distributions and origin servers. AWS WAF is a web application firewall that helps secure your web applications and APIs by blocking requests before they reach your servers. For more information, see Accelerate and protect your websites using CloudFront and AWS WAF and Guidelines for Implementing AWS WAF.

To enable AWS WAF protections, you can:

- Use one-click protection in the CloudFront console. One-click protection creates an AWS WAF web access control list (web ACL), configures rules to protect your servers from common web threats, and attaches the web ACL to the CloudFront distribution for you. The topics in this section assume the use of one-click protections.
- Use a preconfigured web ACL (access control list) that you create in the AWS WAF console, or by using the AWS WAF APIs. For more information, see Web access control lists (ACLs) in the *AWS WAF Developer Guide* and AssociateWebACL in the *AWS WAF API Reference*

You can enable AWS WAF when you:

- Create a distribution
- Use the **Security** dashboard to edit the security settings of an existing distribution

When you use one-click protection, CloudFront applies an AWS recommended set of protections that:

- Block IP addresses from potential threats based on Amazon internal threat intelligence.
- Protect against the most common vulnerabilities found in web applications as described in the OWASP Top 10.
- Defend against malicious actors discovering application vulnerabilities.

Important

You must enable AWS WAF if you want to view security metrics in the CloudFront **Security** dashboard. Without AWS WAF enabled, you can only use the **Security** dashboard to enable AWS WAF or configure CloudFront geographic restrictions. For more information about

the dashboard, see Manage AWS WAF security protections in the CloudFront security dashboard, later in this section.

Topics

- Enable AWS WAF for distributions
- Manage AWS WAF security protections in the CloudFront security dashboard
- Set up rate limiting
- Disable AWS WAF security protections

Enable AWS WAF for distributions

You can enable AWS WAF when you create a distribution, or you can enable security protections for an existing access control list (ACL).

If you enable AWS WAF for your CloudFront distribution, you can also enable bot control and configure security protection by bot category.

Topics

- Enable AWS WAF for a new distribution
- Use an existing web ACL
- Enable bot control
- Configure protection by bot category

Enable AWS WAF for a new distribution

The following procedure shows you how to enable AWS WAF when you create a new CloudFront distribution.

To enable AWS WAF for a new distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, and then choose **Create distribution**.
3. As needed, follow the steps in Create a distribution.

4. In the **Web Application Firewall** section, choose **Edit**, then choose **Enable security protections**.
5. Complete the following fields:

   - **Use monitor mode** – You enable monitor mode when you want to first collect data to test how protection will work. When you enable monitor mode, requests aren't blocked if the protections were active. Instead, monitor mode collects data about requests that would be blocked if the protections were active. When you're ready to begin blocking, you can enable blocking on the **Security** page.
   - **Additional protections** – Choose any options that you want to enable. If you enable rate limiting, see the section called “Set up rate limiting” for more information.
   - **Price estimate** – You can open the section to display a field where you enter a different number of requests/month and see a new estimate.
6. Review the remaining distribution settings, then choose **Create distribution**.

After you create a distribution, CloudFront creates a **Security** dashboard. You can use this dashboard to disable or enable AWS WAF. If you haven't enabled AWS WAF yet, the charts and graphs in the dashboard remain blank.

Use an existing web ACL

If you have an existing web ACL, you can use it instead of the protection offered by AWS WAF.

To use an existing AWS WAF configuration

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Do one of the following:

   a. Choose **Create distribution** and follow the steps in Create a distribution, then return to this topic.

   b. Choose an existing configuration, and then choose the **Security** tab.
3. In the **Web Application Firewall (WAF)** section, choose **Edit**, then **Enable security protections**.
4. Choose **Use existing WAF configuration**. This option appears only if you have web ACLs configured.

5. Choose your existing web ACL from the **Choose a web ACL** table.
6. Review the remaining distribution settings, and then choose **Create distribution**.

Enable bot control

If you enable AWS WAF for your CloudFront distribution, you can view bot requests for a given time range under the security dashboard in the CloudFront console. You can also enable or disable bot control here.

You incur charges when you enable bot control. The security dashboard provides a cost estimate.

If you enable bot control, the security dashboard displays bot traffic by each bot type and category. If you disable bot control, bot traffic is displayed based on request sampling.

To enable bot control

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, then choose the distribution that you want to change.
3. Choose the **Security** tab.
4. Scroll down to the **Bot requests for a given time range** section and choose **Enable Bot Control**.
5. In the **Bot Control** dialog box, under **Configuration**, select the **Enable Bot Control for common bots** check box.
6. Choose **Save changes**.

Configure protection by bot category

When you enable bot control, you can configure how each unverified bot is handled per bot category. For example, you can set an HTTP library bot to **Monitor mode** and assign a **Challenge** to a link checker.

> **Note**
>
> Bots that are known by AWS to be common and verifiable, such as known search engine crawlers, aren't subject to the actions you set here. Bot control confirms that validated bots come from the source that they claim before marking them as verified.

To configure protection for a bot category

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, then choose the distribution that you want to change.
3. Choose the **Security** tab.
4. In the **Requests by bot category** chart, point to any of the items in the **Unverified bot action** column and choose the pencil icon to edit it.
5. Open the resulting list and choose one of the following:
   - **Block**
   - **Allow**
   - **Monitor mode**
   - **CAPTCHA**
   - **Challenge**
6. Select the check mark next to the list to save your change.

Manage AWS WAF security protections in the CloudFront security dashboard

CloudFront creates a security dashboard for each of your distributions. You use the dashboards in the CloudFront console. With the dashboards, you can use CloudFront and AWS WAF together in a single location to monitor and manage common security protections for your web applications. The dashboards provide the following tasks and data:

- **Security configuration** – You can enable and disable AWS WAF protections, and see any app-specific protections such as WordPress protections.

- **Security trends** – These include allowed and blocked requests, challenge and CAPTCHA requests, and top attack types. You can see traffic ratios and how they change over time. For example, if all requests increase by 3% but allowed requests increase by 14%, that means you allowed a larger portion of your traffic through in the current period.
- **Bot requests** – You can see how much traffic comes from bots, which types of bots (verified vs non-verified), and how the percentage allocations of bot types (verified vs non-verified) change over time. For more information about enabling bot control, see Enable bot control.
- **Request logs** – Log data can help answer questions about security trends or bot requests. You can search your logs without writing queries, and view aggregate charts to help determine if a filtered set of logs is primarily being driven by a subset of HTTP methods, IP addresses, URI paths, or countries. You can hover over values in the charts and block IP addresses and countries. For more information, see Enable AWS WAF logs.
- **Geographic restriction management** – CloudFront and AWS WAF provide geographic restriction features. CloudFront provides geographic restrictions for free, but metrics for CloudFront geographic restrictions aren't displayed in the security dashboard. To see request metrics for blocked country requests, you must use AWS WAF geographic restrictions. To do this, hover over a country bar in the security dashboard and block the country. For more information, see Use CloudFront geographic restrictions.
  - The **Block** option might not be available if you previously created a custom AWS WAF rule outside of the CloudFront console to block countries.

Topics

- Prerequisites
- Enable AWS WAF logs

Prerequisites

You must enable AWS WAF if you want to view security metrics in the CloudFront **Security** dashboard. If you don't enable AWS WAF, you can only use the **Security** dashboard to enable AWS WAF or configure CloudFront geographic restrictions.

For more information about enabling AWS WAF, see Enable AWS WAF for distributions.

Enable AWS WAF logs

AWS WAF log data can help you isolate specific traffic patterns. For example, logs can show you where certain traffic comes from or what it does.

If you enable AWS WAF logging to CloudWatch, the CloudFront security dashboard queries, aggregates, and displays insights from the CloudWatch logs. We don't charge to use the security dashboard, but CloudWatch pricing applies to logs queried through the dashboard. For more information, see Amazon CloudWatch Pricing.

To enable logs

1. Enter your expected request volume in the **Number of requests/month** box to estimate the costs of enabling logs.
2. Select the **Enable AWS WAF logs** check box.
3. Choose **Enable**.

CloudFront creates a CloudWatch logs group and updates your AWS WAF configuration to begin logging to CloudWatch. On first use, log data can take several minutes to appear. The **Requests** section of the chart lists each request. Below the individual requests, the bar charts aggregate data by HTTP method, top URI paths, top IP addresses, and top countries. The charts can help you find patterns. For example, you may see a disproportionate volume of requests from a single IP address, or data from a country that you haven't previously seen in your logs. You can filter requests based on **Country**, **Host Header**, and other attributes to help find unwanted traffic. After you identify that traffic, hover over an individual request or chart item and block an IP address or country.

> **Note**
>
> Displayed metrics are based on web ACL. Therefore, if you associate the same web ACL to multiple distributions, you will see all metrics for your web ACL, not only the AWS WAF requests that are processed for that distribution.

Set up rate limiting

Rate limiting is one of the recommendations you may receive when configuring security protections.

CloudFront always enables rate limiting in monitor mode. When monitor mode is enabled, CloudFront captures metrics that tell you if the rate you configured in the **Rate limiting** field has been exceeded, how often, and by how much.

After you save the distribution, CloudFront starts to collect data based on the number in the **Rate limiting** field.

You can enable or manage the rate limiting settings in the **Security - Web Application Firewall (WAF)** section on the **Security** tab of any CloudFront distribution.

> **Note**
>
> The **Rate limiting** option only appears in the CloudFront console if you specified a non-S3 custom origin for your distribution. Otherwise, you will only see the **Core protections** enabled for the distribution. For more information about origin types, see Use various origins with CloudFront distributions.

To set up rate limiting

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, and then choose the distribution that you want to change.
3. Choose the **Security** tab.
4. In the **Security – Web Application Firewall (WAF)** section, choose **Edit**.
5. Under **Additional protections**, select **Rate limiting**. You can optionally change the rate limit. When you have fine-tuned the rate, choose **Save changes**.
6. In the **Security – Web Application Firewall (WAF)** section, next to **Rate limiting**, you can choose **Monitor mode** and then choose **Enable blocking** to deactivate monitor mode. CloudFront will start to block requests that exceed the specified rate limit.

For more information about enabling AWS WAF and rate limiting, see the Introducing CloudFront Security Dashboard, a Unified CDN and Security Experience blog post.

Disable AWS WAF security protections

If your distribution doesn't need AWS WAF security protections, you can disable this feature by using the CloudFront console.

If you previously enabled AWS WAF protection and didn't choose an existing WAF configuration (also known as one-click protection), CloudFront automatically created a web ACL for you. For web ACLs created this way, the CloudFront console will disassociate the resource and delete the web ACL.

Disassociating a web ACL is different from deleting it. Disassociating removes the web ACL from your distribution, but it's not deleted from your AWS account. For more information, see Associating or disassociating a web ACL with an AWS resource in the *AWS WAF, AWS Firewall Manager, and AWS Shield Advanced Developer Guide*.

See the following procedure to disable AWS WAF protections and disassociate the web ACL from your distribution.

To disable AWS WAF security protections in CloudFront

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**, and then choose the distribution that you want to change.
3. Choose the **Security** tab and then choose **Edit**.
4. In the **Web Application Firewall (WAF)** section, choose **Disable AWS WAF protection**.
5. Choose **Save changes**.

Notes

- If you disabled AWS WAF security protection and you still want to delete the web ACL from your AWS account, you can delete it manually. Follow the procedure to delete a web ACL. In the AWS WAF & Shield console, for the **Web ACLs** page, you *must* choose the **Global (CloudFront)** list to find the web ACLs.

- When you delete a distribution from the CloudFront console, CloudFront will attempt to also delete the web ACL if you chose one-click protection. This is best effort and isn't always guaranteed. For more information, see Delete a distribution.

Configure secure access and restrict access to content

CloudFront provides several options for securing content that it delivers. The following are some ways you can use CloudFront to secure and restrict access to content:

- Configure HTTPS connections
- Prevent users in specific geographic locations from accessing content
- Require users to access content using CloudFront signed URLs or signed cookies
- Set up field-level encryption for specific content fields
- Use AWS WAF to control access to your content

You should also implement a DDoS-resilient architecture for your infrastructure and applications. For more information, see AWS Best Practices for DDoS Resiliency.

For additional information, see the following:

- Securing your content delivery with CloudFront
- SIEM on Amazon OpenSearch Service

Topics

- Use HTTPS with CloudFront
- Use alternate domain names and HTTPS
- Mutual TLS authentication with CloudFront (Viewer mTLS)
- Origin mutual TLS with CloudFront
- Serve private content with signed URLs and signed cookies
- Restrict access to an AWS origin
- Restrict access to Application Load Balancers
- Restrict the geographic distribution of your content
- Use field-level encryption to help protect sensitive data

Use HTTPS with CloudFront

You can configure CloudFront to require that viewers use HTTPS so that connections are encrypted when CloudFront communicates with viewers. You also can configure CloudFront to use HTTPS with your origin so that connections are encrypted when CloudFront communicates with your origin.

If you configure CloudFront to require HTTPS both to communicate with viewers and to communicate with your origin, here's what happens when CloudFront receives a request:

1. A viewer submits an HTTPS request to CloudFront. There's some SSL/TLS negotiation here between the viewer and CloudFront. In the end, the viewer submits the request in an encrypted format.
2. If the CloudFront edge location contains a cached response, CloudFront encrypts the response and returns it to the viewer, and the viewer decrypts it.
3. If the CloudFront edge location doesn't contain a cached response, CloudFront performs SSL/TLS negotiation with your origin and, when the negotiation is complete, forwards the request to your origin in an encrypted format.
4. Your origin decrypts the request, processes it (generates a response), encrypts the response, and returns the response to CloudFront.
5. CloudFront decrypts the response, re-encrypts it, and forwards it to the viewer. CloudFront also caches the response in the edge location so that it's available the next time it's requested.
6. The viewer decrypts the response.

The process works basically the same way whether your origin is an Amazon S3 bucket, MediaStore, or a custom origin such as an HTTP/S server.

> **Note**
>
> To help thwart SSL renegotiation-type attacks, CloudFront does not support renegotiation for viewer and origin requests.

Alternatively, you can turn on mutual authentication for your CloudFront distribution. For more information, see Mutual TLS authentication with CloudFront (Viewer mTLS).

For information about how to require HTTPS between viewers and CloudFront, and between CloudFront and your origin, see the following topics.

Topics

- Require HTTPS for communication between viewers and CloudFront
- Require HTTPS for communication between CloudFront and your custom origin
- Require HTTPS for communication between CloudFront and your Amazon S3 origin
- Supported protocols and ciphers between viewers and CloudFront
- Supported protocols and ciphers between CloudFront and the origin

Require HTTPS for communication between viewers and CloudFront

You can configure one or more cache behaviors in your CloudFront distribution to require HTTPS for communication between viewers and CloudFront. You also can configure one or more cache behaviors to allow both HTTP and HTTPS, so that CloudFront requires HTTPS for some objects but not for others. The configuration steps depend on which domain name you're using in object URLs:

- If you're using the domain name that CloudFront assigned to your distribution, such as d111111abcdef8.cloudfront.net, you change the **Viewer Protocol Policy** setting for one or more cache behaviors to require HTTPS communication. In that configuration, CloudFront provides the SSL/TLS certificate.

  To change the value of **Viewer Protocol Policy** by using the CloudFront console, see the procedure later in this section.

  For information about how to use the CloudFront API to change the value of the `ViewerProtocolPolicy` element, see UpdateDistribution in the *Amazon CloudFront API Reference*.
- If you're using your own domain name, such as example.com, you need to change several CloudFront settings. You also need to use an SSL/TLS certificate provided by AWS Certificate Manager (ACM), or import a certificate from a third-party certificate authority into ACM or the IAM certificate store. For more information, see Use alternate domain names and HTTPS.

> **Note**
>
> If you want to make sure that the objects that viewers get from CloudFront were encrypted when CloudFront got them from your origin, always use HTTPS between CloudFront and your origin. If you recently changed from HTTP to HTTPS between CloudFront and your origin, we recommend that you invalidate objects in CloudFront edge locations. CloudFront will return an object to a viewer regardless of whether the protocol used by the viewer (HTTP or HTTPS) matches the protocol that CloudFront used to get the object. For more information about removing or replacing objects in a distribution, see Add, remove, or replace content that CloudFront distributes.

Require HTTPS for viewers

To require HTTPS between viewers and CloudFront for one or more cache behaviors, perform the following procedure.

To configure CloudFront to require HTTPS between viewers and CloudFront

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the top pane of the CloudFront console, choose the ID for the distribution that you want to update.
3. On the **Behaviors** tab, select the cache behavior that you want to update, and then choose **Edit**.
4. Specify one of the following values for **Viewer protocol policy**:

   Redirect HTTP to HTTPS

   Viewers can use both protocols. HTTP `GET` and `HEAD` requests are automatically redirected to HTTPS requests. CloudFront returns HTTP status code 301 (Moved Permanently) along with the new HTTPS URL. The viewer then resubmits the request to CloudFront using the HTTPS URL.

   > **Important**
   >
   > If you send `POST`, `PUT`, `DELETE`, `OPTIONS`, or `PATCH` over HTTP with an HTTP to HTTPS cache behavior and a request protocol version of HTTP 1.1 or above,

CloudFront redirects the request to an HTTPS location with an HTTP status code 307 (Temporary Redirect). This guarantees that the request is sent again to the new location using the same method and body payload.
If you send `POST`, `PUT`, `DELETE`, `OPTIONS`, or `PATCH` requests over HTTP to HTTPS cache behavior with a request protocol version below HTTP 1.1, CloudFront returns an HTTP status code 403 (Forbidden).

When a viewer makes an HTTP request that is redirected to an HTTPS request, CloudFront charges for both requests. For the HTTP request, the charge is only for the request and for the headers that CloudFront returns to the viewer. For the HTTPS request, the charge is for the request, and for the headers and the object that are returned by your origin.

HTTPS only

Viewers can access your content only if they're using HTTPS. If a viewer sends an HTTP request instead of an HTTPS request, CloudFront returns HTTP status code 403 (Forbidden) and does not return the object.

5. Choose **Save changes**.
6. Repeat steps 3 through 5 for each additional cache behavior that you want to require HTTPS for between viewers and CloudFront.
7. Confirm the following before you use the updated configuration in a production environment:

   - The path pattern in each cache behavior applies only to the requests that you want viewers to use HTTPS for.
   - The cache behaviors are listed in the order that you want CloudFront to evaluate them in. For more information, see Path pattern.
   - The cache behaviors are routing requests to the correct origins.

Require HTTPS for communication between CloudFront and your custom origin

You can require HTTPS for communication between CloudFront and your origin.

> **Note**
>
> If your origin is an Amazon S3 bucket that's configured as a website endpoint, you can't configure CloudFront to use HTTPS with your origin because Amazon S3 doesn't support HTTPS for website endpoints.

To require HTTPS between CloudFront and your origin, follow the procedures in this topic to do the following:

1. In your distribution, change the **Origin Protocol Policy** setting for the origin.
2. Install an SSL/TLS certificate on your origin server (this isn't required when you use an Amazon S3 origin or certain other AWS origins).

Topics

- Require HTTPS for custom origins
- Install an SSL/TLS certificate on your custom origin

Require HTTPS for custom origins

The following procedure explains how to configure CloudFront to use HTTPS to communicate with an Elastic Load Balancing load balancer, an Amazon EC2 instance, or another custom origin. For information about using the CloudFront API to update a distribution, see UpdateDistribution in the *Amazon CloudFront API Reference*.

To configure CloudFront to require HTTPS between CloudFront and your custom origin

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the top pane of the CloudFront console, choose the ID for the distribution that you want to update.
3. On the **Origins** tab, select the origin that you want to update, and then choose **Edit**.
4. Update the following settings:

   Origin Protocol Policy

   Change the **Origin Protocol Policy** for the applicable origins in your distribution:

- **HTTPS Only** – CloudFront uses only HTTPS to communicate with your custom origin.
- **Match Viewer** – CloudFront communicates with your custom origin using HTTP or HTTPS, depending on the protocol of the viewer request. For example, if you choose **Match Viewer** for **Origin Protocol Policy** and the viewer uses HTTPS to request an object from CloudFront, CloudFront also uses HTTPS to forward the request to your origin.

  Choose **Match Viewer** only if you specify **Redirect HTTP to HTTPS** or **HTTPS Only** for **Viewer Protocol Policy**.

  CloudFront caches the object only once even if viewers make requests using both HTTP and HTTPS protocols.

Origin SSL Protocols

Choose the **Origin SSL Protocols** for the applicable origins in your distribution. The SSLv3 protocol is less secure, so we recommend that you choose SSLv3 only if your origin doesn't support TLSv1 or later. The TLSv1 handshake is both backwards and forwards compatible with SSLv3, but TLSv1.1 and later are not. When you choose SSLv3, CloudFront *only* sends SSLv3 handshake requests.

5. Choose **Save changes**.
6. Repeat steps 3 through 5 for each additional origin that you want to require HTTPS for between CloudFront and your custom origin.
7. Confirm the following before you use the updated configuration in a production environment:

   - The path pattern in each cache behavior applies only to the requests that you want viewers to use HTTPS for.
   - The cache behaviors are listed in the order that you want CloudFront to evaluate them in. For more information, see Path pattern.
   - The cache behaviors are routing requests to the origins that you changed the **Origin Protocol Policy** for.

Install an SSL/TLS certificate on your custom origin

You can use an SSL/TLS certificate from the following sources on your custom origin:

- If your origin is an Elastic Load Balancing load balancer, you can use a certificate provided by AWS Certificate Manager (ACM). You also can use a certificate that is signed by a trusted third-party certificate authority and imported into ACM.

- For origins other than Elastic Load Balancing load balancers, you must use a certificate that is signed by a trusted third-party certificate authority (CA), for example, Comodo, DigiCert, or Symantec.

The certificate returned from the origin must include one of the following domain names:

- The domain name in the origin's **Origin domain** field (the `DomainName` field in the CloudFront API).
- The domain name in the `Host` header, if the cache behavior is configured to forward the `Host` header to the origin.

When CloudFront uses HTTPS to communicate with your origin, CloudFront verifies that the certificate was issued by a trusted certificate authority. CloudFront supports the same certificate authorities that Mozilla does. For the current list, see Mozilla Included CA Certificate List. You can't use a self-signed certificate for HTTPS communication between CloudFront and your origin.

> **Important**
>
> If the origin server returns an expired certificate, an invalid certificate, or a self-signed certificate, or if the origin server returns the certificate chain in the wrong order, CloudFront drops the TCP connection, returns HTTP status code 502 (Bad Gateway) to the viewer, and sets the `X-Cache` header to `Error from cloudfront`. Also, if the full chain of certificates, including the intermediate certificate, is not present, CloudFront drops the TCP connection.

Require HTTPS for communication between CloudFront and your Amazon S3 origin

When your origin is an Amazon S3 bucket, your options for using HTTPS for communications with CloudFront depend on how you're using the bucket. If your Amazon S3 bucket is configured as a website endpoint, you can't configure CloudFront to use HTTPS to communicate with your origin because Amazon S3 doesn't support HTTPS connections in that configuration.

When your origin is an Amazon S3 bucket that supports HTTPS communication, CloudFront forwards requests to S3 by using the protocol that viewers used to submit the requests. The default setting for the Protocol (custom origins only) setting is **Match Viewer** and can't be

changed. However, if you enable origin access control (OAC) for your Amazon S3 origin, the communication used between CloudFront and Amazon S3 depends on your settings. For more information, see Create a new origin access control.

If you want to require HTTPS for communication between CloudFront and Amazon S3, you must change the value of **Viewer Protocol Policy** to **Redirect HTTP to HTTPS** or **HTTPS Only**. The procedure later in this section explains how to use the CloudFront console to change **Viewer Protocol Policy**. For information about using the CloudFront API to update the `ViewerProtocolPolicy` element for a distribution, see UpdateDistribution in the *Amazon CloudFront API Reference*.

When you use HTTPS with an Amazon S3 bucket that supports HTTPS communication, Amazon S3 provides the SSL/TLS certificate, so you don't have to.

Require HTTPS for an Amazon S3 origin

The following procedure shows you how to configure CloudFront to require HTTPS to your Amazon S3 origin.

To configure CloudFront to require HTTPS to your Amazon S3 origin

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the top pane of the CloudFront console, choose the ID for the distribution that you want to update.
3. On the **Behaviors** tab, choose the cache behavior that you want to update, and then choose **Edit**.
4. Specify one of the following values for **Viewer Protocol Policy**:

   Redirect HTTP to HTTPS

   Viewers can use both protocols, but HTTP requests are automatically redirected to HTTPS requests. CloudFront returns HTTP status code 301 (Moved Permanently) along with the new HTTPS URL. The viewer then resubmits the request to CloudFront using the HTTPS URL.

> **Important**
>
> CloudFront doesn't redirect `DELETE`, `OPTIONS`, `PATCH`, `POST`, or `PUT` requests from HTTP to HTTPS. If you configure a cache behavior to redirect to HTTPS, CloudFront responds to HTTP `DELETE`, `OPTIONS`, `PATCH`, `POST`, or `PUT` requests for that cache behavior with HTTP status code 403 (Forbidden).

When a viewer makes an HTTP request that is redirected to an HTTPS request, CloudFront charges for both requests. For the HTTP request, the charge is only for the request and for the headers that CloudFront returns to the viewer. For the HTTPS request, the charge is for the request, and for the headers and the object returned by your origin.

HTTPS Only

Viewers can access your content only if they're using HTTPS. If a viewer sends an HTTP request instead of an HTTPS request, CloudFront returns HTTP status code 403 (Forbidden) and does not return the object.

5. Choose **Yes, Edit**.
6. Repeat steps 3 through 5 for each additional cache behavior that you want to require HTTPS for between viewers and CloudFront, and between CloudFront and S3.
7. Confirm the following before you use the updated configuration in a production environment:
   - The path pattern in each cache behavior applies only to the requests that you want viewers to use HTTPS for.
   - The cache behaviors are listed in the order that you want CloudFront to evaluate them in. For more information, see Path pattern.
   - The cache behaviors are routing requests to the correct origins.

Supported protocols and ciphers between viewers and CloudFront

When you require HTTPS between viewers and your CloudFront distribution, you must choose a security policy, which determines the following settings:

- The minimum SSL/TLS protocol that CloudFront uses to communicate with viewers.
- The ciphers that CloudFront can use to encrypt the communication with viewers.

To choose a security policy, specify the applicable value for Security policy (minimum SSL/TLS version). The following table lists the protocols and ciphers that CloudFront can use for each security policy.

A viewer must support at least one of the supported ciphers to establish an HTTPS connection with CloudFront. CloudFront chooses a cipher in the listed order from among the ciphers that the viewer supports. See also OpenSSL, s2n, and RFC cipher names.

	Security policy								
	SSLv3	**TLSv1**	**TLSv1_6**	**TLSv1.016**	**TLSv1.018**	**TLSv1.019**	**TLSv1.021**	**TLSv1.025**	**TLSv1.3_2025**
Supported SSL/TLS protocols									
TLSv1.3	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLSv1.2	♦	♦	♦	♦	♦	♦	♦	♦	
TLSv1.1	♦	♦	♦	♦					
TLSv1	♦	♦	♦						
SSLv3	♦								
Supported TLSv1.3 ciphers									
TLS_AES_128_GCM_SHA256	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_AES_256_GCM_SHA384	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_CHACHA20_POLY1305_SHA256	♦	♦	♦	♦	♦	♦	♦		♦
Supported ECDSA ciphers									

	Security policy								
	SSLv3	TLSv1	TLSv1_6	TLSv1.016	TLSv1.018	TLSv1.019	TLSv1.021	TLSv1.025	TLSv1.3_2025
ECDHE-ECDSA-AES128-GCM-SHA256	♦	♦	♦	♦	♦	♦	♦	♦	
ECDHE-ECDSA-AES128-SHA256	♦	♦	♦	♦	♦	♦			
ECDHE-ECDSA-AES128-SHA	♦	♦	♦	♦					
ECDHE-ECDSA-AES256-GCM-SHA384	♦	♦	♦	♦	♦	♦	♦	♦	
ECDHE-ECDSA-CHACHA20-POLY1305	♦	♦	♦	♦	♦	♦	♦		
ECDHE-ECDSA-AES256-SHA384	♦	♦	♦	♦	♦	♦			
ECDHE-ECDSA-AES256-SHA	♦	♦	♦	♦					
Supported RSA ciphers									
ECDHE-RSA-AES128-GCM-SHA256	♦	♦	♦	♦	♦	♦	♦	♦	
ECDHE-RSA-AES128-SHA256	♦	♦	♦	♦	♦	♦			
ECDHE-RSA-AES128-SHA	♦	♦	♦	♦					

	Security policy								
	SSLv3	TLSv1	TLSv1_6	TLSv1.016	TLSv1.018	TLSv1.019	TLSv1.021	TLSv1.025	TLSv1.3_2025
ECDHE-RSA-AES256-GCM-SHA384	♦	♦	♦	♦	♦	♦	♦	♦	
ECDHE-RSA-CHACHA20-POLY1305	♦	♦	♦	♦	♦	♦	♦		
ECDHE-RSA-AES256-SHA384	♦	♦	♦	♦	♦	♦			
ECDHE-RSA-AES256-SHA	♦	♦	♦	♦					
AES128-GCM-SHA256	♦	♦	♦	♦	♦				
AES256-GCM-SHA384	♦	♦	♦	♦	♦				
AES128-SHA256	♦	♦	♦	♦	♦				
AES256-SHA	♦	♦	♦	♦					
AES128-SHA	♦	♦	♦	♦					
DES-CBC3-SHA	♦	♦							
RC4-MD5	♦								

OpenSSL, s2n, and RFC cipher names

OpenSSL and s2n use different names for ciphers than the TLS standards use (RFC 2246, RFC 4346, RFC 5246, and RFC 8446). The following table maps the OpenSSL and s2n names to the RFC name for each cipher.

CloudFront supports both classical and quantum-safe key exchanges. For classical key exchanges using elliptic curves, CloudFront supports the following:

- `prime256v1`
- `X25519`
- `secp384r1`

For quantum-safe key exchanges, CloudFront supports the following:

- `X25519MLKEM768`
- `SecP256r1MLKEM768`

> **Note**
>
> Quantum-safe key exchanges are only supported with TLS 1.3. TLS 1.2 and earlier versions do not support quantum-safe key exchanges.

For more information, see the following topics:

- Post-Quantum Cryptography
- Cryptography algorithms and AWS services
- Hybrid key exchange in TLS 1.3

For more information about certificate requirements for CloudFront, see Requirements for using SSL/TLS certificates with CloudFront.

OpenSSL and s2n cipher name	RFC cipher name
Supported TLSv1.3 ciphers	
TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256
Supported ECDSA ciphers	

OpenSSL and s2n cipher name	RFC cipher name
ECDHE-ECDSA-AES128-GCM-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
ECDHE-ECDSA-AES128-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
ECDHE-ECDSA-AES128-SHA	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
ECDHE-ECDSA-AES256-GCM-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
ECDHE-ECDSA-CHACHA20-POLY1305	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
ECDHE-ECDSA-AES256-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
ECDHE-ECDSA-AES256-SHA	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
Supported RSA ciphers	
ECDHE-RSA-AES128-GCM-SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
ECDHE-RSA-AES128-SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
ECDHE-RSA-AES128-SHA	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
ECDHE-RSA-AES256-GCM-SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
ECDHE-RSA-CHACHA20-POLY1305	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
ECDHE-RSA-AES256-SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

OpenSSL and s2n cipher name	RFC cipher name
ECDHE-RSA-AES256-SHA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
AES128-GCM-SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256
AES256-GCM-SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384
AES128-SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256
AES256-SHA	TLS_RSA_WITH_AES_256_CBC_SHA
AES128-SHA	TLS_RSA_WITH_AES_128_CBC_SHA
DES-CBC3-SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
RC4-MD5	TLS_RSA_WITH_RC4_128_MD5

Supported signature schemes between viewers and CloudFront

CloudFront supports the following signature schemes for connections between viewers and CloudFront.

	Security policy								
Signature schemes	**SSLv3**	**TLSv1**	**TLSv1_6**	**TLSv1.016**	**TLSv1.018**	**TLSv1.019**	**TLSv1.021**	**TLSv1.025**	**TLSv1.3_2025**
TLS_SIGNATURE_SCHEME_RSA_PSS_PSS_SHA256	◆	◆	◆	◆	◆	◆	◆	◆	◆
TLS_SIGNATURE_SCHEME_RSA_PSS_PSS_SHA384	◆	◆	◆	◆	◆	◆	◆	◆	◆

	Security policy								
Signature schemes	**SSLv3**	**TLSv1**	**TLSv1_6**	**TLSv1.016**	**TLSv1.018**	**TLSv1.019**	**TLSv1.021**	**TLSv1.025**	**TLSv1.3_2025**
TLS_SIGNATURE_SCHEME_RSA_PSS_PSS_SHA512	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA384	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA512	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA256	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA384	♦	♦	♦	♦	♦	♦	♦	♦	♦

	Security policy								
Signature schemes	**SSLv3**	**TLSv1**	**TLSv1_6**	**TLSv1.016**	**TLSv1.018**	**TLSv1.019**	**TLSv1.021**	**TLSv1.025**	**TLSv1.3_2025**
TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA512	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA224	♦	♦	♦	♦	♦	♦	♦		
TLS_SIGNATURE_SCHEME_ECDSA_SHA256	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_ECDSA_SHA384	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_ECDSA_SHA512	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_ECDSA_SHA224	♦	♦	♦	♦	♦	♦	♦		
TLS_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256	♦	♦	♦	♦	♦	♦	♦	♦	♦

	Security policy								
Signature schemes	**SSLv3**	**TLSv1**	**TLSv1_6**	**TLSv1.016**	**TLSv1.018**	**TLSv1.019**	**TLSv1.021**	**TLSv1.025**	**TLSv1.3_2025**
TLS_SIGNATURE_SCHEME_ECDSA_SECP384R1_SHA384	♦	♦	♦	♦	♦	♦	♦	♦	♦
TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA1	♦	♦	♦	♦					
TLS_SIGNATURE_SCHEME_ECDSA_SHA1	♦	♦	♦	♦					

Supported protocols and ciphers between CloudFront and the origin

If you choose to require HTTPS between CloudFront and your origin, you can decide which SSL/TLS protocol to allow for the secure connection, and CloudFront can connect to the origin using any of the ECDSA or RSA ciphers listed in the following table. Your origin must support at least one of these ciphers for CloudFront to establish an HTTPS connection to your origin.

OpenSSL and s2n use different names for ciphers than the TLS standards use (RFC 2246, RFC 4346, RFC 5246, and RFC 8446). The following table includes the OpenSSL and s2n name, and the RFC name, for each cipher.

For ciphers with elliptic curve key exchange algorithms, CloudFront supports the following elliptic curves:

- prime256v1
- secp384r1
- X25519

OpenSSL and s2n cipher name	RFC cipher name
Supported ECDSA ciphers	
ECDHE-ECDSA-AES256-GCM-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
ECDHE-ECDSA-AES256-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
ECDHE-ECDSA-AES256-SHA	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
ECDHE-ECDSA-AES128-GCM-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
ECDHE-ECDSA-AES128-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
ECDHE-ECDSA-AES128-SHA	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
Supported RSA ciphers	
ECDHE-RSA-AES256-GCM-SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
ECDHE-RSA-AES256-SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
ECDHE-RSA-AES256-SHA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
ECDHE-RSA-AES128-GCM-SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
ECDHE-RSA-AES128-SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
ECDHE-RSA-AES128-SHA	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
AES256-SHA	TLS_RSA_WITH_AES_256_CBC_SHA

OpenSSL and s2n cipher name	RFC cipher name
AES128-SHA	TLS_RSA_WITH_AES_128_CBC_SHA
DES-CBC3-SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
RC4-MD5	TLS_RSA_WITH_RC4_128_MD5

Supported signature schemes between CloudFront and the origin

CloudFront supports the following signature schemes for connections between CloudFront and the origin.

- TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA256
- TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA384
- TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA512
- TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA224
- TLS_SIGNATURE_SCHEME_ECDSA_SHA256
- TLS_SIGNATURE_SCHEME_ECDSA_SHA384
- TLS_SIGNATURE_SCHEME_ECDSA_SHA512
- TLS_SIGNATURE_SCHEME_ECDSA_SHA224
- TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA1
- TLS_SIGNATURE_SCHEME_ECDSA_SHA1

Use alternate domain names and HTTPS

If you want to use your own domain name in the URLs for your files (for example, `https://www.example.com/image.jpg`) and you want your viewers to use HTTPS, you must complete the steps in the following topics. (If you use the default CloudFront distribution domain name in your URLs, for example, `https://d111111abcdef8.cloudfront.net/image.jpg`, follow the guidance in the following topic instead: Require HTTPS for communication between viewers and CloudFront.)

> **Important**
>
> When you add a certificate to your distribution, CloudFront immediately propagates the certificate to all of its edge locations. As new edge locations become available, CloudFront propagates the certificate to those locations, too. You can't restrict the edge locations that CloudFront propagates the certificates to.

Topics

- Choose how CloudFront serves HTTPS requests
- Requirements for using SSL/TLS certificates with CloudFront
- Quotas on using SSL/TLS certificates with CloudFront (HTTPS between viewers and CloudFront only)
- Configure alternate domain names and HTTPS
- Determine the size of the public key in an SSL/TLS RSA certificate
- Increase the quotas for SSL/TLS certificates
- Rotate SSL/TLS certificates
- Revert from a custom SSL/TLS certificate to the default CloudFront certificate
- Switch from a custom SSL/TLS certificate with dedicated IP addresses to SNI

Choose how CloudFront serves HTTPS requests

If you want your viewers to use HTTPS and to use alternate domain names for your files, choose one of the following options for how CloudFront serves HTTPS requests:

- Use Server Name Indication (SNI) – Recommended
- Use a dedicated IP address in each edge location

This section explains how each option works.

Use SNI to serve HTTPS requests (works for most clients)

Server Name Indication (SNI) is an extension to the TLS protocol that is supported by browsers and clients released after 2010. If you configure CloudFront to serve HTTPS requests using SNI,

CloudFront associates your alternate domain name with an IP address for each edge location. When a viewer submits an HTTPS request for your content, DNS routes the request to the IP address for the correct edge location. The IP address to your domain name is determined during the SSL/TLS handshake negotiation; the IP address isn't dedicated to your distribution.

The SSL/TLS negotiation occurs early in the process of establishing an HTTPS connection. If CloudFront can't immediately determine which domain the request is for, it drops the connection. When a viewer that supports SNI submits an HTTPS request for your content, here's what happens:

1. The viewer automatically gets the domain name from the request URL and adds it to the SNI extension of the TLS client hello message.
2. When CloudFront receives the TLS client hello, it uses the domain name in the SNI extension to find the matching CloudFront distribution and sends back the associated TLS certificate.
3. The viewer and CloudFront perform SSL/TLS negotiation.
4. CloudFront returns the requested content to the viewer.

For a current list of the browsers that support SNI, see the Wikipedia entry Server Name Indication.

If you want to use SNI but some of your users' browsers don't support SNI, you have several options:

- Configure CloudFront to serve HTTPS requests by using dedicated IP addresses instead of SNI. For more information, see Use a dedicated IP address to serve HTTPS requests (works for all clients).
- Use the CloudFront SSL/TLS certificate instead of a custom certificate. This requires that you use the CloudFront domain name for your distribution in the URLs for your files, for example, `https://d111111abcdef8.cloudfront.net/logo.png`.

  If you use the default CloudFront certificate, viewers must support the SSL protocol TLSv1 or later. CloudFront doesn't support SSLv3 with the default CloudFront certificate.

  You also must change the SSL/TLS certificate that CloudFront is using from a custom certificate to the default CloudFront certificate:

  - If you haven't used your distribution to distribute your content, you can just change the configuration. For more information, see Update a distribution.
  - If you have used your distribution to distribute your content, you must create a new CloudFront distribution and change the URLs for your files to reduce or eliminate the amount

of time that your content is unavailable. For more information, see Revert from a custom SSL/TLS certificate to the default CloudFront certificate.

- If you can control which browser your users use, have them upgrade their browser to one that supports SNI.
- Use HTTP instead of HTTPS.

Use a dedicated IP address to serve HTTPS requests (works for all clients)

Server Name Indication (SNI) is one way to associate a request with a domain. Another way is to use a dedicated IP address. If you have users who can't upgrade to a browser or client released after 2010, you can use a dedicated IP address to serve HTTPS requests. For a current list of the browsers that support SNI, see the Wikipedia entry Server Name Indication.

> **Important**
>
> If you configure CloudFront to serve HTTPS requests using dedicated IP addresses, you incur an additional monthly charge. The charge begins when you associate your SSL/TLS certificate with a distribution and you enable the distribution. For more information about CloudFront pricing, see Amazon CloudFront Pricing. In addition, see Using the Same Certificate for Multiple CloudFront Distributions.

When you configure CloudFront to serve HTTPS requests by using dedicated IP addresses, CloudFront associates your certificate with a dedicated IP address in each CloudFront edge location. When a viewer submits an HTTPS request for your content, here's what happens:

1. DNS routes the request to the IP address for your distribution in the applicable edge location.
2. If a client request provides the SNI extension in the `ClientHello` message, CloudFront searches for a distribution that is associated with that SNI.
   - If there's a match, CloudFront responds to the request with the SSL/TLS certificate.
   - If there's no match, CloudFront uses the IP address instead to identify your distribution and to determine which SSL/TLS certificate to return to the viewer.
3. The viewer and CloudFront perform SSL/TLS negotiation using your SSL/TLS certificate.
4. CloudFront returns the requested content to the viewer.

This method works for every HTTPS request, regardless of the browser or other viewer that the user is using.

> **Note**
>
> Dedicated IPs are not static IPs and can change over time. The IP address that is returned for the edge location is allocated dynamically from the IP address ranges of the CloudFront edge servers list.
> The IP address ranges for CloudFront edge servers are subject to change. To be notified of IP address changes, subscribe to AWS Public IP Address Changes by using Amazon SNS.

Request permission to use three or more dedicated IP SSL/TLS certificates

If you need permission to permanently associate three or more SSL/TLS dedicated IP certificates with CloudFront, perform the following procedure. For more details about HTTPS requests, see Choose how CloudFront serves HTTPS requests.

> **Note**
>
> This procedure is for using three or more dedicated IP certificates across your CloudFront distributions. The default value is 2. Keep in mind you cannot bind more than one SSL certificate to a distribution.
> You can only associate a single SSL/TLS certificate to a CloudFront distribution at a time. This number is for the total number of dedicated IP SSL certificates you can use across all of your CloudFront distributions.

To request permission to use three or more certificates with a CloudFront distribution

1. Go to the Support Center and create a case.
2. Indicate how many certificates you need permission to use, and describe the circumstances in your request. Your account will be updated as soon as possible.
3. Continue with the next procedure.

Requirements for using SSL/TLS certificates with CloudFront

The requirements for SSL/TLS certificates are described in this topic. They apply to both of the following, except as noted:

- Certificates for using HTTPS between viewers and CloudFront
- Certificates for using HTTPS between CloudFront and your origin

Topics

- Certificate issuer
- AWS Region for AWS Certificate Manager
- Certificate format
- Intermediate certificates
- Key type
- Private key
- Permissions
- Size of the certificate key
- Supported types of certificates
- Certificate expiration date and renewal
- Domain names in the CloudFront distribution and in the certificate
- Minimum SSL/TLS protocol version
- Supported HTTP versions

Certificate issuer

We recommend that you use a public certificate issued by AWS Certificate Manager (ACM). For information about getting a certificate from ACM, see the *AWS Certificate Manager User Guide*. To use an ACM certificate with a CloudFront distribution, make sure you request (or import) the certificate in the US East (N. Virginia) Region (`us-east-1`).

CloudFront supports the same certificate authorities (CAs) as Mozilla, so if you don't use ACM, use a certificate issued by a CA on the Mozilla Included CA Certificate List.

TLS certificates used by the origin that you specified for your CloudFront distribution also need to be issued from the CA on the Mozilla Included CA Certificate List.

For more information about getting and installing a certificate, refer to the documentation for your HTTP server software and to the documentation for the CA.

AWS Region for AWS Certificate Manager

To use a certificate in AWS Certificate Manager (ACM) to require HTTPS between viewers and CloudFront, make sure you request (or import) the certificate in the US East (N. Virginia) Region (`us-east-1`).

If you want to require HTTPS between CloudFront and your origin, and you're using a load balancer in Elastic Load Balancing as your origin, you can request or import the certificate in any AWS Region.

Certificate format

The certificate must be in X.509 PEM format. This is the default format if you're using AWS Certificate Manager.

Intermediate certificates

If you're using a third-party certificate authority (CA), list all of the intermediate certificates in the certificate chain that's in the `.pem` file, beginning with one for the CA that signed the certificate for your domain. Typically, you'll find a file on the CA website that lists intermediate and root certificates in the proper chained order.

> **Important**
>
> Do not include the following: the root certificate, intermediate certificates that are not in the trust path, or your CA's public key certificate.

Here's an example:

```
-----BEGIN CERTIFICATE-----
Intermediate certificate 2
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Intermediate certificate 1
-----END CERTIFICATE-----
```

Key type

CloudFront supports RSA and ECDSA public–private key pairs.

CloudFront supports HTTPS connections to both viewers and origins using RSA and ECDSA certificates. With AWS Certificate Manager (ACM), you can request and import RSA or ECDSA certificates and then associate them with your CloudFront distribution.

For lists of the RSA and ECDSA ciphers supported by CloudFront that you can negotiate in HTTPS connections, see the section called "Supported protocols and ciphers between viewers and CloudFront" and the section called "Supported protocols and ciphers between CloudFront and the origin".

Private key

If you're using a certificate from a third-party certificate authority (CA), note the following:

- The private key must match the public key that is in the certificate.
- The private key must be in PEM format.
- The private key cannot be encrypted with a password.

If AWS Certificate Manager (ACM) provided the certificate, ACM doesn't release the private key. The private key is stored in ACM for use by AWS services that are integrated with ACM.

Permissions

You must have permission to use and import the SSL/TLS certificate. If you're using AWS Certificate Manager (ACM), we recommend that you use AWS Identity and Access Management permissions to restrict access to the certificates. For more information, see Identity and access management in the *AWS Certificate Manager User Guide*.

Size of the certificate key

The certificate key size that CloudFront supports depends on the type of key and certificate.

For RSA certificates:

CloudFront supports 1024-bit, 2048-bit, 3072-bit, and 4096-bit RSA keys. The maximum key length for an RSA certificate that you use with CloudFront is 4096 bits.

Note that ACM issues RSA certificates with up to 2048-bit keys. To use a 3072-bit or 4096-bit RSA certificate, you need to obtain the certificate externally and import it into ACM, after which it will be available for you to use with CloudFront.

For information about how to determine the size of an RSA key, see Determine the size of the public key in an SSL/TLS RSA certificate.

For ECDSA certificates:

CloudFront supports 256-bit and 384-bit keys. To use an ECDSA certificate in ACM to require HTTPS between viewers and CloudFront, use the prime256v1 or secp384r1 elliptic curve.

Supported types of certificates

CloudFront supports all types of certificates issued by a trusted certificate authority.

Certificate expiration date and renewal

If you're using certificates that you get from a third-party certificate authority (CA), you must monitor certificate expiration dates and renew the certificates that you import into AWS Certificate Manager (ACM) or upload to the AWS Identity and Access Management certificate store before they expire.

Important

To avoid certificate expiration issues, renew or reimport your certificate at least 24 hours before the `NotAfter` value of your current certificate. If your certificate expires within 24 hours, request a new certificate from ACM or import a new certificate to ACM. Next, associate the new certificate to the CloudFront distribution.
CloudFront might continue to use the previous certificate while your certificate renewal or reimport is in progress. This is an asynchronous process that can take up to 24 hours before CloudFront shows your changes.

If you're using ACM provided certificates, ACM manages certificate renewals for you. For more information, see Managed renewal in the *AWS Certificate Manager User Guide*.

Domain names in the CloudFront distribution and in the certificate

When you're using a custom origin, the SSL/TLS certificate on your origin includes a domain name in the **Common Name** field, and possibly several more in the **Subject Alternative Names** field. (CloudFront supports wildcard characters in certificate domain names.)

One of the domain names in the certificate must match the domain name that you specify for Origin Domain Name. If no domain name matches, CloudFront returns HTTP status code `502 (Bad Gateway)` to the viewer.

> **Important**
>
> When you add an alternate domain name to a distribution, CloudFront checks that the alternate domain name is covered by the certificate that you've attached. The certificate must cover the alternate domain name in the subject alternative name (SAN) field of the certificate. This means the SAN field must contain an exact match for the alternate domain name, or contain a wildcard at the same level of the alternate domain name that you're adding.
> For more information, see Requirements for using alternate domain names.

Minimum SSL/TLS protocol version

If you're using dedicated IP addresses, set the minimum SSL/TLS protocol version for the connection between viewers and CloudFront by choosing a security policy.

For more information, see Security policy (minimum SSL/TLS version) in the topic All distribution settings reference.

Supported HTTP versions

If you associate one certificate with more than one CloudFront distribution, all the distributions associated with the certificate must use the same option for Supported HTTP versions. You specify this option when you create or update a CloudFront distribution.

Quotas on using SSL/TLS certificates with CloudFront (HTTPS between viewers and CloudFront only)

Note the following quotas on using SSL/TLS certificates with CloudFront. These quotas apply only to the SSL/TLS certificates that you provision by using AWS Certificate Manager (ACM), that you

import into ACM, or upload to the IAM certificate store for HTTPS communication between viewers and CloudFront.

For more information, see Increase the quotas for SSL/TLS certificates.

Maximum number of certificates per CloudFront distribution

You can associate a maximum of one SSL/TLS certificate with each CloudFront distribution.

Maximum number of certificates that you can import into ACM or upload to the IAM certificate store

If you obtained your SSL/TLS certificates from a third-party CA, you must store the certificates in one of the following locations:

- **AWS Certificate Manager** – For the current quota on the number of ACM certificates, see Quotas in the *AWS Certificate Manager User Guide*. The listed quota is a total that includes certificates that you provision by using ACM and certificates that you import into ACM.
- **IAM certificate store** – For the current quota (formerly known as limit) on the number of certificates that you can upload to the IAM certificate store for an AWS account, see IAM and STS Limits in the *IAM User Guide*. You can request a higher quota in the Service Quotas console.

Maximum number of certificates per AWS account (dedicated IP addresses only)

If you want to serve HTTPS requests by using dedicated IP addresses, note the following:

- By default, CloudFront gives you permission to use two certificates with your AWS account, one for everyday use and one for when you need to rotate certificates for multiple distributions.
- If you need more than two custom SSL/TLS certificates for your AWS account, you can request a higher quota in the Service Quotas console.

Use the same certificate for CloudFront distributions that were created by using different AWS accounts

If you're using a third-party CA and you want to use the same certificate with multiple CloudFront distributions that were created by using different AWS accounts, you must import the certificate into ACM or upload it to the IAM certificate store once for each AWS account.

If you're using certificates provided by ACM, you can't configure CloudFront to use certificates that were created by a different AWS account.

Use the same certificate for CloudFront and for other AWS services

If you bought a certificate from a trusted certificate authority such as Comodo, DigiCert, or Symantec, you can use the same certificate for CloudFront and for other AWS services. If you're importing the certificate into ACM, you need to import it only once to use it for multiple AWS services.

If you're using certificates provided by ACM, the certificates are stored in ACM.

Use the same certificate for multiple CloudFront distributions

You can use the same certificate for any or all of the CloudFront distributions that you're using to serve HTTPS requests. Note the following:

- You can use the same certificate both for serving requests using dedicated IP addresses and for serving requests using SNI.
- You can associate only one certificate with each distribution.
- Each distribution must include one or more alternate domain names that also appear in the **Common Name** field or the **Subject Alternative Names** field in the certificate.
- If you're serving HTTPS requests using dedicated IP addresses and you created all of your distributions by using the same AWS account, you can significantly reduce your cost by using the same certificate for all distributions. CloudFront charges for each certificate, not for each distribution.

  For example, suppose you create three distributions by using the same AWS account, and you use the same certificate for all three distributions. You would be charged only one fee for using dedicated IP addresses.

  However, if you're serving HTTPS requests using dedicated IP addresses and using the same certificate to create CloudFront distributions in different AWS accounts, each account is charged the fee for using dedicated IP addresses. For example, if you create three distributions by using three different AWS accounts and you use the same certificate for all three distributions, each account is charged the full fee for using dedicated IP addresses.

Configure alternate domain names and HTTPS

To use alternate domain names in the URLs for your files and to use HTTPS between viewers and CloudFront, perform the applicable procedures.

Topics

- Get an SSL/TLS certificate
- Import an SSL/TLS certificate
- Update your CloudFront distribution

Get an SSL/TLS certificate

Get an SSL/TLS certificate if you don't already have one. For more information, see the applicable documentation:

- To use a certificate provided by AWS Certificate Manager (ACM), see the AWS Certificate Manager User Guide. Then skip to Update your CloudFront distribution.

  > **Note**
  >
  > We recommend that you use ACM to provision, manage, and deploy SSL/TLS certificates on AWS managed resources. You must request an ACM certificate in the US East (N. Virginia) Region.

- To get a certificate from a third-party certificate authority (CA), see the documentation provided by the certificate authority. When you have the certificate, continue with the next procedure.

Import an SSL/TLS certificate

If you got your certificate from a third-party CA, import the certificate into ACM or upload it to the IAM certificate store:

ACM (recommended)

With ACM, you can import third-party certificates from the ACM console, as well as programmatically. For information about importing a certificate to ACM, see Importing Certificates into AWS Certificate Manager in the *AWS Certificate Manager User Guide*. You must import the certificate in the US East (N. Virginia) Region.

IAM certificate store

(Not recommended) Use the following AWS CLI command to upload your third-party certificate to the IAM certificate store.

```
aws iam upload-server-certificate \
```

```
        --server-certificate-name CertificateName \
        --certificate-body file://public_key_certificate_file \
        --private-key file://privatekey.pem \
        --certificate-chain file://certificate_chain_file \
        --path /cloudfront/path/
```

Note the following:

- **AWS account** – You must upload the certificate to the IAM certificate store using the same AWS account that you used to create your CloudFront distribution.
- **--path parameter** – When you upload the certificate to IAM, the value of the `--path` parameter (certificate path) must start with `/cloudfront/`, for example, `/cloudfront/production/` or `/cloudfront/test/`. The path must end with a /.
- **Existing certificates** – You must specify values for the `--server-certificate-name` and `--path` parameters that are different from the values that are associated with existing certificates.
- **Using the CloudFront console** – The value that you specify for the `--server-certificate-name` parameter in the AWS CLI, for example, `myServerCertificate`, appears in the **SSL Certificate** list in the CloudFront console.
- **Using the CloudFront API** – Make note of the alphanumeric string that the AWS CLI returns, for example, `AS1A2M3P4L5E67SIIXR3J`. This is the value that you will specify in the `IAMCertificateId` element. You don't need the IAM ARN, which is also returned by the CLI.

For more information about the AWS CLI, see the AWS Command Line Interface User Guide and the AWS CLI Command Reference.

Update your CloudFront distribution

To update settings for your distribution, perform the following procedure:

To configure your CloudFront distribution for alternate domain names

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the ID for the distribution that you want to update.
3. On the **General** tab, choose **Edit**.
4. Update the following values:

Alternate domain name (CNAME)

Choose **Add item** to add the applicable alternate domain names. Separate domain names with commas, or type each domain name on a new line.

Custom SSL certificate

Select a certificate from the dropdown list.

Up to 100 certificates are listed here. If you have more than 100 certificates and you don't see the certificate that you want to add, you can type a certificate ARN in the field to choose it.

If you uploaded a certificate to the IAM certificate store but it's not listed, and you can't choose it by typing the name in the field, review the procedure Import an SSL/TLS certificate to confirm that you correctly uploaded the certificate.

> **Important**
>
> After you associate your SSL/TLS certificate with your CloudFront distribution, do not delete the certificate from ACM or the IAM certificate store until you remove the certificate from all distributions and all the distributions are deployed.

5. Choose **Save changes**.
6. Configure CloudFront to require HTTPS between viewers and CloudFront:

   a. On the **Behaviors** tab, choose the cache behavior that you want to update, and choose **Edit**.

   b. Specify one of the following values for **Viewer Protocol Policy**:

   Redirect HTTP to HTTPS

   Viewers can use both protocols, but HTTP requests are automatically redirected to HTTPS requests. CloudFront returns HTTP status code `301 (Moved Permanently)` along with the new HTTPS URL. The viewer then resubmits the request to CloudFront using the HTTPS URL.

> **Important**
>
> CloudFront doesn't redirect `DELETE`, `OPTIONS`, `PATCH`, `POST`, or `PUT` requests from HTTP to HTTPS. If you configure a cache behavior to redirect to HTTPS, CloudFront responds to HTTP `DELETE`, `OPTIONS`, `PATCH`, `POST`, or `PUT` requests for that cache behavior with HTTP status code `403 (Forbidden)`.

When a viewer makes an HTTP request that is redirected to an HTTPS request, CloudFront charges for both requests. For the HTTP request, the charge is only for the request and for the headers that CloudFront returns to the viewer. For the HTTPS request, the charge is for the request, and for the headers and the file returned by your origin.

HTTPS Only

Viewers can access your content only if they're using HTTPS. If a viewer sends an HTTP request instead of an HTTPS request, CloudFront returns HTTP status code `403 (Forbidden)` and does not return the file.

c. Choose **Yes, Edit**.

d. Repeat steps a through c for each additional cache behavior that you want to require HTTPS for between viewers and CloudFront.

7. Confirm the following before you use the updated configuration in a production environment:

- The path pattern in each cache behavior applies only to the requests that you want viewers to use HTTPS for.
- The cache behaviors are listed in the order that you want CloudFront to evaluate them in. For more information, see Path pattern.
- The cache behaviors are routing requests to the correct origins.

Determine the size of the public key in an SSL/TLS RSA certificate

When you're using CloudFront alternate domain names and HTTPS, the maximum size of the public key in an SSL/TLS RSA certificate is 4096 bits. (This is the key size, not the number of characters in the public key.) If you use AWS Certificate Manager for your certificates, although ACM supports larger RSA keys, you cannot use the larger keys with CloudFront.

You can determine the size of the RSA public key by running the following OpenSSL command:

```
openssl x509 -in path and filename of SSL/TLS certificate -text -noout
```

Where:

- `-in` specifies the path and file name of your SSL/TLS RSA certificate.
- `-text` causes OpenSSL to display the length of the RSA public key in bits.
- `-noout` prevents OpenSSL from displaying the public key.

Example output:

```
Public-Key: (2048 bit)
```

Increase the quotas for SSL/TLS certificates

There are quotas on the number of SSL/TLS certificates that you can import into AWS Certificate Manager (ACM) or upload to AWS Identity and Access Management (IAM). There also is a quota on the number of SSL/TLS certificates that you can use with an AWS account when you configure CloudFront to serve HTTPS requests by using dedicated IP addresses. However, you can request higher quotas.

Topics

- Increase quota on certificates imported into ACM
- Increase quota on certificates uploaded to IAM
- Increase quota on certificates used with dedicated IP addresses

Increase quota on certificates imported into ACM

For the quota on the number of certificates that you can import into ACM, see Quotas in the *AWS Certificate Manager User Guide*.

To request a higher quota, use the Service Quotas console. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

Increase quota on certificates uploaded to IAM

For the quota (formerly known as limit) on the number of certificates that you can upload to IAM, see IAM and STS Limits in the *IAM User Guide*.

To request a higher quota, use the Service Quotas console. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

Increase quota on certificates used with dedicated IP addresses

For the quota on the number of SSL certificates that you can use for each AWS account when serving HTTPS requests using dedicated IP addresses, see Quotas on SSL certificates.

To request a higher quota, use the Service Quotas console. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

Rotate SSL/TLS certificates

When your SSL/TLS certificates are near expiration, you need to rotate them to make sure the security for your distribution and avoid service disruption for your viewers. You can rotate them in the following ways:

- For SSL/TLS certificates provided by AWS Certificate Manager (ACM), you don't need to rotate them. ACM *automatically* manages certificate renewals for you. For more information, see Managed certificate renewal in the *AWS Certificate Manager User Guide*.
- If you're using a third-party certificate authority and you imported the certificates into ACM (recommended) or uploaded them to the IAM certificate store, you must occasionally replace one certificate with another.

Important

- ACM doesn't manage certificate renewals for certificates that you acquire from third-party certificate authorities and import into ACM.
- If you configured CloudFront to serve HTTPS requests by using dedicated IP addresses, you might incur an additional, pro-rated charge for using one or more additional certificates while you're rotating certificates. We recommend that you update your distributions to minimize the additional charge.

Rotate SSL/TLS certificates

To rotate your certificates, perform the following procedure. Viewers can continue to access your content while you rotate certificates as well as after the process is complete.

To rotate SSL/TLS certificates

1. Increase the quotas for SSL/TLS certificates to determine whether you need permission to use more SSL certificates. If so, request permission and wait until permission is granted before you continue with step 2.
2. Import the new certificate into ACM or upload it to IAM. For more information, see Importing an SSL/TLS Certificate in the *Amazon CloudFront Developer Guide*.
3. (For IAM certificates only) Update your distributions one at a time to use the new certificate. For more information, see Update a distribution.
4. (Optional) Delete the previous certificate from ACM or IAM.

> **Important**
>
> Don't delete an SSL/TLS certificate until you remove it from all distributions and until the status of the distributions that you have updated has changed to `Deployed`.

Revert from a custom SSL/TLS certificate to the default CloudFront certificate

If you configured CloudFront to use HTTPS between viewers and CloudFront, and you configured CloudFront to use a custom SSL/TLS certificate, you can change your configuration to use the default CloudFront SSL/TLS certificate. The process depends on whether you've used your distribution to distribute your content:

- If you have not used your distribution to distribute your content, you can just change the configuration. For more information, see Update a distribution.
- If you have used your distribution to distribute your content, you must create a new CloudFront distribution and change the URLs for your files to reduce or eliminate the amount of time that your content is unavailable. To do that, perform the following procedure.

Revert to default CloudFront certificate

The following procedure shows you how to revert from a custom SSL/TLS certificate to the default CloudFront certificate.

To revert to the default CloudFront certificate

1. Create a new CloudFront distribution with the desired configuration. For **SSL Certificate**, choose **Default CloudFront Certificate (*.cloudfront.net)**.

   For more information, see Create a distribution.

2. For files that you're distributing using CloudFront, update the URLs in your application to use the domain name that CloudFront assigned to the new distribution. For example, change `https://www.example.com/images/logo.png` to `https://d111111abcdef8.cloudfront.net/images/logo.png`.

3. Either delete the distribution that is associated with a custom SSL/TLS certificate, or update the distribution to change the value of **SSL Certificate** to **Default CloudFront Certificate (*.cloudfront.net)**. For more information, see Update a distribution.

   > **Important**
   >
   > Until you complete this step, AWS continues to charge you for using a custom SSL/TLS certificate.

4. (Optional) Delete your custom SSL/TLS certificate.

   a. Run the AWS CLI command `list-server-certificates` to get the certificate ID of the certificate that you want to delete. For more information, see list-server-certificates in the *AWS CLI Command Reference*.

   b. Run the AWS CLI command `delete-server-certificate` to delete the certificate. For more information, see delete-server-certificate in the *AWS CLI Command Reference*.

Switch from a custom SSL/TLS certificate with dedicated IP addresses to SNI

If you configured CloudFront to use a custom SSL/TLS certificate with dedicated IP addresses, you can switch to using a custom SSL/TLS certificate with SNI instead and eliminate the charge that is associated with dedicated IP addresses.

> **Important**
>
> This update to your CloudFront configuration has no effect on viewers that support SNI. Viewers can access your content before and after the change, as well as while the change is propagating to CloudFront edge locations. Viewers that don't support SNI can't access your content after the change. For more information, see Choose how CloudFront serves HTTPS requests.

Switch from a custom certificate to SNI

The following procedure shows you how to switch from a custom SSL/TLS certificate with dedicated IP addresses to SNI.

To switch from a custom SSL/TLS certificate with dedicated IP addresses to SNI

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the ID of the distribution that you want to view or update.
3. Choose **Distribution Settings**.
4. On the **General** tab, choose **Edit**.
5. Under **Custom SSL certification – *optional***, deselect **Legacy clients support**.
6. Choose **Yes, Edit**.

Mutual TLS authentication with CloudFront (Viewer mTLS)

Mutual TLS Authentication (Mutual Transport Layer Security Authentication — mTLS) is a security protocol that extends standard TLS authentication by requiring bidirectional certificate-based authentication, where both client and server must prove their identity before establishing a

secure connection. Using mutual TLS, you can make sure that only clients presenting trusted TLS certificates gain access to your CloudFront distributions.

How it works

In a standard TLS handshake, only the server presents a certificate to prove its identity to the client. With mutual TLS, the authentication process becomes bidirectional. When a client attempts to connect to your CloudFront distribution, CloudFront requests a client certificate during the TLS handshake. The client must present a valid X.509 certificate that CloudFront validates against your configured trust store before establishing the secure connection.

CloudFront performs this certificate validation at AWS edge locations, offloading the authentication complexity from your origin servers while maintaining CloudFront global performance benefits. You can configure mTLS in three modes:

- **Required mode** (default) — CloudFront validates the client certificate against a trust store. If validation fails or no certificate is presented, CloudFront denies the connection. Use required mode when every client must authenticate with a valid certificate.
- **Optional mode** — CloudFront validates the client certificate if one is presented, but allows connections without a certificate. Certificate metadata is available in Connection Functions and HTTP headers for your origin to make authorization decisions. Use optional mode when you support both authenticated and unauthenticated clients.
- **Passthrough mode** — CloudFront does not validate the client certificate against a trust store. CloudFront only validates that the client possesses the corresponding private key. It forwards the certificate to your origin as HTTP headers for your origin to perform validation. No trust store is required and no caching occurs. Use passthrough mode when you have existing mTLS implementations at your origin.

For more information about these validation modes, see Additional validation modes.

Use cases

Mutual TLS authentication with CloudFront addresses several critical security scenarios where traditional authentication methods are insufficient:

- **Device authentication with content caching** - You can authenticate gaming consoles, IoT devices, or corporate hardware before allowing access to firmware updates, game downloads,

or internal resources. Each device contains a unique certificate that proves its authenticity while benefiting from CloudFront caching capabilities.

- **API-to-API authentication** - You can secure machine-to-machine communication between trusted business partners, payment systems, or micro-services. Certificate-based authentication eliminates the need for shared secrets or API keys while providing strong identity verification for automated data exchanges.

Topics

- Trust stores and certificate management
- Enable mutual TLS for CloudFront distributions
- Associate a CloudFront Connection Function
- Configuring additional settings
- Viewer mTLS headers for cache policies and forwarded to origin
- Certificate revocation
- Helper methods for mutual TLS
- Additional validation modes
- Connection logs for mutual TLS

Trust stores and certificate management

Creating and configuring a trust store is a mandatory requirement for implementing mutual TLS authentication with CloudFront. Trust stores contain the Certificate Authority (CA) certificates that CloudFront uses to validate client certificates during the authentication process.

What is a trust store?

A trust store is a repository of CA certificates that CloudFront uses to validate client certificates during mutual TLS authentication. Trust stores contain the root and intermediate CA certificates that form the chain of trust for authenticating client certificates.

When you implement mutual TLS with CloudFront, the trust store defines which Certificate Authorities you trust to issue valid client certificates. CloudFront validates each client certificate against your trust store during the TLS handshake. Only clients presenting certificates that chain to one of the CAs in your trust store will be authenticated successfully.

Trust stores in CloudFront are account-level resources that you can associate with multiple distributions. With this, you can maintain consistent certificate validation policies across your entire CloudFront deployment while simplifying CA certificate management.

Certificate Authority support

CloudFront supports certificates issued by both AWS Private Certificate Authority and third-party private Certificate Authorities. With this flexibility, you can use your existing certificate infrastructure or leverage AWS managed certificate services based on your organizational requirements.

- **AWS Private Certificate Authority:** You can use certificates issued by AWS Private CA, which provides a managed private certificate authority service. This integration simplifies certificate lifecycle management and provides seamless integration with other AWS services.
- **Third-party private Certificate Authorities:** You can also use certificates from your existing private Certificate Authority infrastructure, including enterprise CAs or other third-party certificate providers. With this, you can maintain your current certificate management processes while adding CloudFront mTLS capabilities.

Certificate requirements and specifications

Trust stores have specific requirements for the CA certificates they contain:

CA certificate format requirements

- **Format:** PEM (Privacy Enhanced Mail) format
- **Content boundaries:** Certificates must be enclosed within the -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- boundaries
- **Comments:** Must be preceded by a # character and cannot contain any - characters
- **Line breaks:** No blank lines are allowed between certificates

Supported certificate specifications

- **Certificate type:** X.509v3
- **Public key types:**
  - RSA 2048, RSA 3072, RSA 4096
  - ECDSA: secp256r1, secp384r1, secp521r1

- **Signature algorithms:**
  - SHA256, SHA384, SHA512 with RSA
  - SHA256, SHA384, SHA512 with EC
  - SHA256, SHA384, SHA512 with RSASSA-PSS with MGF1

Example certificate bundle format

Multiple certificates (PEM-encoded):

```
# Root CA Certificate
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAKoK/OvD/XqiMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQwHhcNMTcwNzEyMTU0NzQ4WhcNMjcwNzEwMTU0NzQ4WjBF
MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAuuExKvY1xzHFylsHiuowqpmzs7rEcuuylOuEszpFp+BtXh0ZuEtts9LP
-----END CERTIFICATE-----
# Intermediate CA Certificate
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAKoK/OvD/XqjMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQwHhcNMTcwNzEyMTU0NzQ4WhcNMjcwNzEwMTU0NzQ4WjBF
MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAuuExKvY1xzHFylsHiuowqpmzs7rEcuuylOuEszpFp+BtXh0ZuEtts9LP
-----END CERTIFICATE-----
```

Create a trust store

Before creating a trust store, you must upload your CA certificate bundle in PEM format to an Amazon S3 bucket. The certificate bundle should contain all the trusted root and intermediate CA certificates needed to validate your client certificates.

The CA certificate bundle is only read once from S3 when creating a trust store. If future changes are made to the CA certificate bundle, then the trust store will have to be manually updated. No sync is maintained between the trust store and the S3 CA certificate bundle.

Prerequisites

- A certificate bundle from your Certificate Authority (CA) uploaded to an Amazon S3 bucket

- The necessary permissions to create CloudFront resources

To create a trust store (Console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Trust stores**.
3. Choose **Create trust store**.
4. For **Trust store name**, enter a name for your trust store.
5. For **Certificate authority (CA) bundle**, enter the Amazon S3 path to your PEM-format CA certificate bundle.
6. Choose **Create trust store**.

To create a trust store (AWS CLI)

```
aws cloudfront create-trust-store \
  --name MyTrustStore \
  --ca-certificates-bundle-source '{"CaCertificatesBundleS3Location":{"Bucket":"my-bucket","Key":"ca-bundle.pem","Region":"bucket-region"}}' \
  --tags Items=[{Key=Environment,Value=Production}]
```

Associate trust store with distributions

After creating a trust store, you must associate it with a CloudFront distribution to enable mutual TLS authentication.

Prerequisites

- An existing CloudFront distribution with HTTPS-only viewer protocol policy enabled and HTTP3 support disabled.

To associate a trust store (Console)

There are two avenues to associate a trust store within the CloudFront console: through the trust store details page or through the distribution settings page.

Associating a trust store through the trust store details page:

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Trust stores**.
3. Choose the name of the trust store you want to associate.
4. Choose **Associate to distribution**.
5. Configure the available Viewer mTLS options:

   - **Client certificate validation mode:** Choose between Required and Optional mode. In required mode, all clients are required to present certificates. In optional mode, clients that present certificates are validated, while clients that do not present certificates are permitted access.
   - **Advertise trust store CA names:** Choose whether to advertise the CA names in your trust store to clients during TLS handshake.
   - **Ignore certificate expiration date:** Choose whether to allow connections with expired certificates (other validation criteria still apply).
   - **Connection Function:** An optional Connection Function can be associated to allow/deny connections based on other custom criteria.
6. Select one or more distributions to associate with the trust store. Only distributions with HTTP3 disabled and with HTTPS-only cache behaviors can support Viewer mTLS.
7. Choose **Associate**.

Associating a trust store through the distribution settings page:

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Select the distribution that you want to associate
3. Under the **General** tab, within the **Settings** container, choose **Edit** in the top right corner
4. Scroll down to the bottom of the page, within the **Connectivity** container, toggle the **Viewer mTLS** switch on
5. Configure the available Viewer mTLS options:

   - **Client certificate validation mode:** Choose between Required and Optional mode. In required mode, all clients are required to present certificates. In optional mode, clients that present certificates are validated, while clients that do not present certificates are permitted access.

- **Advertise trust store CA names:** Choose whether to advertise the CA names in your trust store to clients during TLS handshake.
- **Ignore certificate expiration date:** Choose whether to allow connections with expired certificates (other validation criteria still apply).
- **Connection Function:** An optional Connection Function can be associated to allow/deny connections based on other custom criteria.

6. Choose **Save changes** in the bottom right corner.

To associate a trust store (AWS CLI)

Trust stores can be associated to distributions by using the DistributionConfig.ViewerMtlsConfig property. This means you first need to fetch the distribution config and then provide the ViewerMtlsConfig in a subsequent UpdateDistribution request.

```
// First fetch the distribution
aws cloudfront get-distribution {DISTRIBUTION_ID}

// Update the distribution config, for example:
Distribution config, file://distConf.json:
{
  ...other fields,
  ViewerMtlsConfig: {
    Mode: 'required',
    TrustStoreConfig: {
        AdvertiseTrustStoreCaNames: false,
        IgnoreCertificateExpiry: true,
        TrustStoreId: {TRUST_STORE_ID}
    }
  }
}

aws cloudfront update-distribution \
   --id {DISTRIBUTION_ID} \
   --if-match {ETAG} \
   --distribution-config file://distConf.json
```

Manage trust stores

View trust store details

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Trust stores**.
3. Choose the name of the trust store to view its details page.

The details page shows:

- Trust store name and ID
- Number of CA certificates
- Creation date and last modified date
- Associated distributions
- Tags

Modify a trust store

To replace the CA certificate bundle:

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Trust stores**.
3. Choose the name of the trust store.
4. Choose **Actions**, then **Edit**.
5. For **Certificate authority (CA) bundle**, enter the Amazon S3 location of the updated CA bundle PEM file.
6. Choose **Update trust store**.

Delete a trust store

Prerequisites: You must first disassociate the trust store from all CloudFront distributions.

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. In the navigation pane, choose **Trust stores**.
3. Choose the name of the trust store.
4. Choose **Delete trust store**.
5. Choose **Delete** to confirm.

Next steps

After creating and associating your trust store with a CloudFront distribution, you can proceed to enable mutual TLS authentication on your distribution and configure additional settings such as forwarding certificate headers to your origins. For detailed instructions on enabling mTLS on your distributions, see Enable mutual TLS for CloudFront distributions.

Enable mutual TLS for CloudFront distributions

Prerequisites and requirements

CloudFront mutual TLS verify mode requires all clients to present valid certificates during the TLS handshake and rejects connections without valid certificates. Before enabling mutual TLS on a CloudFront distribution, make sure you have:

- Created a trust store with your Certificate Authority certificates
- Associated the trust store with your CloudFront distribution
- Ensured all distribution cache behaviors use an HTTPS-only viewer protocol policy
- Ensured your distribution is using HTTP/2 (the default setting, Viewer mTLS is not supported on HTTP/3)

> **Note**
>
> Mutual TLS authentication requires HTTPS connections between viewers and CloudFront. You cannot enable mTLS on a distribution with any cache behaviors that support HTTP connections.

Enable mutual TLS (Console)

For new distributions

Viewer mTLS cannot be configured in the process of creating a new distribution in the CloudFront console. First create the distribution by any means (console, CLI, API), then edit the distribution settings to enable Viewer mTLS per the existing distributions instructions below.

For existing distributions

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. From the distribution list, select the distribution you want to modify.
3. Make sure the Viewer protocol policy is set to **Redirect HTTP to HTTPS** or **HTTPS Only** for all cache behaviors. (You can choose the **Cache behaviors** tab to view and update any cache behaviors with HTTP protocol policies.)
4. Choose the **General** tab.
5. In the **Settings** section, choose **Edit**.
6. In the **Connectivity** section, find **Viewer mutual authentication (mTLS)**.
7. Toggle **Enable mutual authentication** to On.
8. For **Client certificate validation mode**, select **Required** (all clients must present certificates) or **Optional** (clients can optionally present certificates).
9. For **Trust store**, select your previously created trust store.
10. (Optional) Toggle **Advertise trust store CA names** if you want CloudFront to send CA names to clients during the TLS handshake.
11. (Optional) Toggle **Ignore certificate expiration date** if you want to allow connections with expired certificates.
12. Choose **Save changes**.

Enable mutual TLS (AWS CLI)

For new distributions

The following example shows how to create a distribution configuration file (distribution-config.json) that includes mTLS settings:

```
{
  "CallerReference": "cli-example-1",
  "Origins": {
    "Quantity": 1,
    "Items": [
      {
        "Id": "my-origin",
        "DomainName": "example.com",
        "CustomOriginConfig": {
          "HTTPPort": 80,
          "HTTPSPort": 443,
          "OriginProtocolPolicy": "https-only"
        }
      }
    ]
  },
  "DefaultCacheBehavior": {
    "TargetOriginId": "my-origin",
    "ViewerProtocolPolicy": "https-only",
    "MinTTL": 0,
    "ForwardedValues": {
      "QueryString": false,
      "Cookies": {
        "Forward": "none"
      }
    }
  },
  "ViewerCertificate": {
    "CloudFrontDefaultCertificate": true
  },
  "ViewerMtlsConfig": {
    "Mode": "required",
    "TrustStoreConfig": {
        "TrustStoreId": {TRUST_STORE_ID},
        "AdvertiseTrustStoreCaNames": true,
        "IgnoreCertificateExpiry": true
    }
  },
  "Enabled": true
}
```

Create the distribution with mTLS enabled using the following example command:

```
aws cloudfront create-distribution --distribution-config file://distribution-
config.json
```

For existing distributions

Get the current distribution configuration using the following example command:

```
aws cloudfront get-distribution-config --id E1A2B3C4D5E6F7 --output json > dist-
config.json
```

Edit the file to add mTLS settings. Add the following example section to your distribution configuration:

```
"ViewerMtlsConfig": {
    "Mode": "required",
    "TrustStoreConfig": {
        "TrustStoreId": {TRUST_STORE_ID},
        "AdvertiseTrustStoreCaNames": true,
        "IgnoreCertificateExpiry": true
    }
}
```

Remove the ETag field from the file but save its value separately.

Update the distribution with the new configuration using the following example command:

```
aws cloudfront update-distribution \
    --id E1A2B3C4D5E6F7 \
    --if-match YOUR-ETAG-VALUE \
    --distribution-config file://dist-config.json
```

Viewer protocol policies

When using mutual TLS, all distribution cache behaviors must be configured with an HTTPS-only viewer protocol policy:

- **Redirect HTTP to HTTPS** - Redirects HTTP requests to HTTPS before performing certificate validation.
- **HTTPS Only** - Only accepts HTTPS requests and performs certificate validation.

> **Note**
>
> The HTTP and HTTPS viewer protocol policy is not supported with mutual TLS since HTTP connections cannot perform certificate validation.

Next steps

After enabling Viewer TLS on your CloudFront distribution, you can associate Connection Functions to implement custom certificate validation logic. Connection Functions allow you to extend the built-in mTLS authentication capabilities with custom validation rules, certificate revocation checking, and logging. For details on creating and associating Connection Functions, see Associate a CloudFront Connection Function.

Associate a CloudFront Connection Function

CloudFront Connection Functions allow you to implement custom certificate validation logic during TLS handshakes, providing extensions to the built-in mTLS authentication capabilities.

What are Connection Functions?

Connection Functions are JavaScript functions that run during the TLS handshake after client certificates have been validated. The validated client certificate is passed to the Connection Function at which point the Connection Function can make an additional determination on whether to grant access or not. For detailed information about Connection Functions, see Customize with CloudFront Connection Functions.

How Connection Functions work with mTLS

When a client attempts to establish an mTLS connection to your CloudFront distribution, the following sequence occurs:

1. Client initiates TLS handshake with CloudFront edge location.
2. CloudFront requests and receives client certificate.
3. CloudFront performs standard certificate validation against trust store.
4. If the certificate passes standard validation, CloudFront invokes your Connection Function. If **IgnoreCertificateExpiry** is enabled within your **ViewerMtlsConfig**, then your expired—but otherwise valid—certificates are also passed to the Connection Function. If the client certificates are invalid, Connection Functions will not be invoked.

5. Your Connection Function receives parsed certificate information and connection details.
6. Your function makes an allow/deny decision based on custom logic.
7. CloudFront completes or terminates the TLS connection based on your decision.

Connection Functions are invoked for both verify mode and optional mode (when clients present certificates).

Create a Connection Function

You can create Connection Functions using the CloudFront console or AWS CLI.

To create a Connection Function (Console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Functions**.
3. Choose the **Connection Functions** tab and choose **Create Connection Function**.
4. Enter a function name that is unique within your AWS account.
5. Choose **Continue**.
6. In the function editor, write your JavaScript code for certificate validation. The function handler must call either allow or deny.
7. Optional: a KeyValue store can be associated to the Connection Function to implement revocation control.
8. Choose **Save changes**.

To create a Connection Function (AWS CLI)

The following example shows how to create a Connection Function:

Write your function code in a separate file, for example code.js:

```
function connectionHandler(connection) {
  connection.allow();
}
```

```
aws cloudfront create-connection-function \
  --name "certificate-validator" \
```

```
  --connection-function-config '{
      "Comment": "Client certificate validation function",
      "Runtime": "cloudfront-js-2.0"
  }' \
  --connection-function-code fileb://code.js
```

Connection Function code structure

Connection Functions implement connectionHandler function that receives a connection object containing certificate and connection information. Your function must use either `connection.allow()` or `connection.deny()` to make a decision about the connection.

Basic Connection Function example

The following example shows a simple Connection Function that verifies the subject field of client certificates:

```
function connectionHandler(connection) {
    // Only process if a certificate was presented
    if (!connection.clientCertificate) {
        console.log("No certificate presented");
        connection.deny();
    }

    // Check the subject field for specific organization
    const subject = connection.clientCertificate.certificates.leaf.subject;
    if (!subject.includes("O=ExampleCorp")) {
        console.log("Certificate not from authorized organization");
       connection.deny();
    } else {
        // All checks passed
        console.log("Certificate validation passed");
        connection.allow();
    }
}
```

The full specification of client certificate properties available on the connection object are available here:

```
{
  "connectionId": "Fdb-Eb7L9gVn2cFakz7wWyBJIDAD4-oNO6g8r3vXDV132BtnIVtqDA==", // Unique
 identifier for this TLS connection
```

```
 "clientIp": "203.0.113.42", // IP address of the connecting client (IPv4 or IPv6)
 "clientCertificate": {
   "certificates": {
     "leaf": {
       "serialNumber": "4a:3f:5c:92:d1:e8:7b:6c", // Unique serial number assigned by
the issuing CA (hexadecimal)
       "issuer": "CN=Example Corp Intermediate CA,O=Example Corp,C=US", //
Distinguished Name (DN) of the certificate authority that issued this certificate.
       "subject": "CN=client.example.com,O=Example Corp,C=US", // Distinguished Name
(DN) of the certificate holder.
       "validity": {
         "notBefore": "2024-01-15T00:00:00Z", // Certificate validity start date (ISO
8601 format)
         "notAfter": "2025-01-14T23:59:59Z"   // Certificate expiration date (ISO 8601
format)
       },
       "sha256Fingerprint": "a1b2c3d4e5f6...abc123def456", // SHA-256 hash of the
certificate (64 hex characters)
       "ocspEndpoint": "http://ocsp.example.com", // OCSP responder URL from the
certificate's AIA extension.
       "san": "DNS:client.example.com,DNS:api.example.com", // Subject Alternative
Names.
       "spki":
"30820122300d06092a864886f70d0101010500038201 0f003082010a0282010100...", // Subject
Public Key Info (DER-encoded, hex).
       "sanHex":
"3032820f636c69656e742e6578616d706c652e636f6d820f6170692e6578616d706c652e636f6d..." //
Subject Alternative Names (DER-encoded, hex).
     },
     "intermediates": [
       {
         "serialNumber": "00:d4:04:a5:fd:cb:20:37:54", // Serial number of the
intermediate certificate.
         "issuer": "CN=Example Corp Root CA,O=Example Corp,C=US", // Issuer of the
intermediate certificate.
         "subject": "CN=Example Corp Intermediate CA,O=Example Corp,C=US", // Subject
of the intermediate certificate.
         "validity": {
           "notBefore": "2020-01-01T00:00:00Z",
           "notAfter": "2030-01-01T23:59:59Z"
         },
         "sha256Fingerprint": "b2c3d4e5f6a1...def456abc123",
         "ocspEndpoint": "http://ocsp.example.com/intermediate",
         "san": "DNS:ca.example.com",
```

```
          "spki":
 "30820122300d06092a864886f70d0101010500038201 0f003082010a0282010100b389dd513b...",
          "sanHex": "3015820d63612e6578616d706c652e636f6d..."
        }
      ]
    }
  }
}
```

Associate a Connection Function

After creating your Connection Function, you must publish it to the LIVE stage and associate it with your distribution.

To publish and associate a Connection Function (Console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Functions**
3. Choose the **Connection Functions** tab and select your Connection Function.
4. Choose **Publish** to move it to the LIVE stage.
5. Choose **Add association** in the associated distributions table below the publishing section.
6. Select the distribution with Viewer mTLS enabled that you wish to associate.

Alternatively published Connection Functions can also be associated from the distribution details page.

1. Navigate to the console home page where all your distributions are listed.
2. Select the distribution that you wish to associate.
3. Choose the **General** tab.
4. In the **Settings** section, choose **Edit**.
5. In the **Connectivity** section, find **Viewer mutual authentication (mTLS)**.
6. For **Connection Function**, select your function.
7. Choose **Save changes**.

To associate a Connection Function (AWS CLI)

The following example shows how to associate a Connection Function with a distribution:

```
// DistributionConfig:
{
   ...other settings,
    "ConnectionFunctionAssociation": {
        "Id": "cf_30c2CV2elHwCoInb3LtcaUJkZeD"
    }
}
```

Use cases for Connection Functions

Connection Functions enable several advanced mTLS use cases:

- **Certificate attribute validation** - Verify specific fields in client certificates like organizational unit requirements or subject alternative name patterns.
- **Certificate revocation checking** - Implement custom certificate revocation checking using KeyValueStore to store revoked certificate serial numbers.
- **IP-based certificate policies** - Apply different certificate policies based on client IP addresses or geographic restrictions.
- **Multi-tenant validation** - Implement tenant-specific validation rules where different certificate requirements apply based on hostnames or certificate attributes.

> **Note**
>
> Connection Functions run once per client connection during the TLS handshake.
> Connection Functions can only allow or deny connections, not modify HTTP requests/responses.
> Only LIVE stage functions (published) can be associated with distributions.
> Each distribution can have at most one Connection Function.

Next steps

After associating a Connection Function with your CloudFront distribution, you can configure optional settings to customize the behavior of your mTLS implementation. For detailed instructions on configuring additional settings such as an optional client certificate validation mode, see Configuring additional settings.

Configuring additional settings

After enabling basic mutual TLS authentication, you can configure additional settings to customize the authentication behavior for specific use cases and requirements.

Certificate Authority advertisement

The AdvertiseTrustStoreCaNames field controls whether CloudFront sends the list of trusted CA names to clients during the TLS handshake, helping clients select the appropriate certificate.

To configure CA advertisement (Console)

1. In your distribution settings, navigate to the **General** tab, choose **Edit**.
2. Scroll to the **Viewer mutual authentication (mTLS)** section within the **Connectivity** container.
3. Select or de-select the **Advertise trust store CA names** checkbox.
4. Choose **Save changes**.

To configure CA advertisement (AWS CLI)

The following example shows how to enable CA advertisement:

```
"ViewerMtlsConfig": {
   "Mode": "required", // or "optional"
   "TrustStoreConfig": {
      "AdvertiseTrustStoreCaNames": true,
      ...other settings
   }
}
```

Certificate expiration handling

The IgnoreCertificateExpiry property determines how CloudFront responds to expired client certificates. By default, CloudFront rejects expired client certificates, but you can configure it to

accept them when necessary. This is typically enabled for devices with expired certificates that cannot be readily updated.

To configure certificate expiration handling (Console)

1. In your distribution settings, navigate to **General** tab, choose **Edit**.
2. Scroll to the **Viewer mutual authentication (mTLS)** section of the **Connectivity** container.
3. Select or deselect the **Ignore certificate expiration date** checkbox.
4. Choose **Save changes**.

To configure certificate expiration handling (AWS CLI)

The following example shows how to ignore certificate expiration:

```
"ViewerMtlsConfig": {
  "Mode": "required", // or "optional"
  "TrustStoreConfig": {
     "IgnoreCertificateExpiry": false,
     ...other settings
  }
}
```

> **Note**
>
> **IgnoreCertificateExpiry** only applies to the certificates Validity dates. All other certificate validation checks still apply (chain of trust, signature validation).

Next steps

After configuring additional settings, you can set up header forwarding to pass certificate information to your origins, implement certificate revocation using Connection Functions and KeyValueStore, and enable connection logs for monitoring. For details on forwarding certificate information to origins, see Forward Headers to origins.

Viewer mTLS headers for cache policies and forwarded to origin

When using mutual TLS authentication, CloudFront can extract information from client certificates and forward it to your origins as HTTP headers. This allows your origin servers to access certificate details without implementing certificate validation logic.

The following headers are available for creating cache behaviors:

Header name	Description	Example value	Supported Modes
CloudFront-Viewer-Cert-Serial-Number	Hexadecimal representation of the certificate serial number	4a:3f:5c:92:d1:e8:7b:6c	Required, Optional, Passthrough
CloudFront-Viewer-Cert-Issuer	RFC2253 string representation of the issuer's distinguished name (DN)	CN=rootcamtls.com, OU=rootCA,O=mTLS,L=Seattle, ST=Washington,C=US	Required, Optional, Passthrough
CloudFront-Viewer-Cert-Subject	RFC2253 string representation of the subject's distinguished name (DN)	CN=client_.com,OU=client-3, O=mTLS,ST=Washington,C=US	Required, Optional, Passthrough
CloudFront-Viewer-Cert-Present	Either 1 (present) or 0 (not present) indicating whether the certificate is present. This value is always 1 in Required mode.	1	Required, Optional, Passthrough
CloudFront-Viewer-Cert-Sha256	The SHA256 hash of the client certificate	01fbf94ef f5569753420c349f49adbfd80af527537781	Required, Optional, Passthrough

Header name	Description	Example value	Supported Modes
		6e3ab1fb3 71b29cb586	

For origin requests, two additional headers are supplied, in addition to the headers above made available for cache behaviors. Due to potential header size, CloudFront-Viewer-Cert-Pem header is not exposed to edge functions (Lambda@Edge or CloudFront Functions) and is only forwarded to the origin.

Header name	Description	Example value	Supported Modes
CloudFront-Viewer-Cert-Validity	ISO8601 format of the notBefore and notAfter date	CloudFront-Viewer-Cert-Validity: NotBefore=2023-09-21T01:50:17Z;NotAfter=2024-09-20T01:50:17Z	Required, Optional, Passthrough
CloudFront-Viewer-Cert-Pem	URL-encoded PEM format of the leaf certificate	CloudFront-Viewer-Cert-Pem: -----BEGIN%20CERTIFICATE-----%0AMIIG<...reduced...>NmrUlw%0A-----END%20CERTIFICATE-----%0A	Required, Optional
Client-Cert	Leaf certificate	Client-Cert: :MIIG<...reduced...>NmrUlw:	Passthrough

Header name	Description	Example value	Supported Modes
Client-Cert-Chain	Non-leaf certificate chain	Client-Cert-Chain: :MIIG<... reduced.. .>NmrUlw:	Passthrough

Configure header forwarding

Console

In verify mode, CloudFront automatically adds the CloudFront-Viewer-Cert-* headers to all viewer requests. To forward these headers to your origin:

1. From the main list distributions page, select your distribution with viewer mTLS enabled and go to the **Behaviors** tab
2. Select the cache behavior and choose **Edit**
3. In the **Origin request policy** section, choose **Create policy** or select an existing policy
4. Make sure the following headers are included in the origin request policy:
   - CloudFront-Viewer-Cert-Serial-Number
   - CloudFront-Viewer-Cert-Issuer
   - CloudFront-Viewer-Cert-Subject
   - CloudFront-Viewer-Cert-Present
   - CloudFront-Viewer-Cert-Sha256
   - CloudFront-Viewer-Cert-Validity
   - CloudFront-Viewer-Cert-Pem
5. Choose **Create** (for new policies) or **Save changes** (for existing policies)
6. Select the policy within your cache behavior and save changes

Using AWS CLI

The following example shows how to create an origin request policy that includes mTLS headers for verify mode:

```
aws cloudfront create-origin-request-policy \
  --origin-request-policy-config '{
    "Name": "MTLSHeadersPolicy",
    "HeadersConfig": {
      "HeaderBehavior": "whitelist",
      "Headers": {
        "Quantity": 5,
        "Items": [
          "CloudFront-Viewer-Cert-Serial-Number",
          "CloudFront-Viewer-Cert-Issuer",
          "CloudFront-Viewer-Cert-Subject",
          "CloudFront-Viewer-Cert-Validity",
          "CloudFront-Viewer-Cert-Pem"
        ]
      }
    },
    "CookiesConfig": {
      "CookieBehavior": "none"
    },
    "QueryStringsConfig": {
      "QueryStringBehavior": "none"
    }
  }'
```

Header processing considerations

When working with certificate headers, consider these best practices:

- **Header validation:** Verify certificate header values at your origin as an additional security measure
- **Header size limits:** The PEM certificate headers can be large, make sure your origin server can handle them
- **Cache considerations:** Using certificate headers in your cache key increases cache fragmentation
- **Cross-origin requests:** If your application uses CORS, then you might need to configure it to allow the certificate headers

Next steps

After configuring header forwarding, you can implement certificate revocation checking using CloudFront Connection Functions and KeyValueStore. For details on implementing revocation checks, see Certificate revocation.

Certificate revocation

When certificates need to be revoked — due to compromise, policy changes, or terminated relationships — you need a mechanism to reject those certificates during the mTLS handshake. CloudFront provides two native approaches to certificate revocation, and you can combine them for layered control.

- **OCSP (Online Certificate Status Protocol)** — CloudFront queries your Certificate Authority's OCSP responder in real time to check whether a client certificate has been revoked. Enable OCSP on your trust store and CloudFront handles the validation automatically during the TLS handshake. OCSP results are also visible in Connection Functions, giving you programmatic access to revocation status for custom decision-making.
- **CloudFront Functions and KeyValueStore** — You maintain a list of revoked certificate serial numbers in a CloudFront KeyValueStore. A Connection Function queries the KeyValueStore during the TLS handshake and allows or denies the connection. This gives you full control over revocation data, update timing, and custom logic like grace periods or IP-based exceptions.

Comparison: OCSP versus CloudFront Functions with KeyValueStore

	OCSP	CloudFront Functions + KeyValueStore
Data source	Certificate Authority's OCSP responder	You manage the revocation list
Update mechanism	Real-time query to CA	You push updates to KeyValueStore
Custom logic	Available by using Connection Functions	Built into your function code

	OCSP	CloudFront Functions + KeyValueStore
External dependency	Requires CA OCSP responder availability	No external dependency
Best for	CAs that maintain OCSP responders; real-time CA-authoritative status	Self-managed revocation; custom policies; CAs without OCSP support

You can use both approaches together. Enable OCSP for CA-authoritative revocation checking, then use a Connection Function to layer additional logic on top of the OCSP result — for example, allowing revoked certificates from trusted IP ranges during a grace period.

OCSP (Online Certificate Status Protocol)

OCSP is a real-time protocol that checks a certificate's revocation status directly with the Certificate Authority (CA). During the certificate signing process, the CA embeds an OCSP responder URL in the certificate. When a client presents their certificate during the mTLS handshake, CloudFront sends an OCSP request to the embedded responder URL and acts on the response: valid certificates proceed, revoked certificates are terminated.

CloudFront validates the entire certificate chain — the leaf certificate and up to three intermediate certificates — each against their respective OCSP responder URLs. Certificates in the trust store are treated as trust anchors and are excluded from OCSP validation. This applies to any certificate in the trust store, whether it is a root CA or an intermediate CA. When you place an intermediate CA in your trust store, it becomes a trust anchor and chain validation terminates at that certificate. Only the certificates below it in the chain are OCSP-validated.

Enable OCSP

Enable OCSP on your trust store. When enabled, CloudFront automatically performs OCSP validation for any client certificate that contains an OCSP responder URL in its Authority Information Access (AIA) extension. Once OCSP is enabled, every certificate in the client certificate chain below the trust anchor must have an OCSP URL. If any certificate below the trust anchor doesn't contain an OCSP URL, CloudFront does not establish the connection. Certificates in the trust store (trust anchors) are exempt from this requirement.

CloudFront caches OCSP responses at the edge to reduce round-trip time and protect against OCSP responder downtime. OCSP responses are cached for approximately 30 minutes, and updated revocation status might take up to 30 minutes to be reflected.

OCSP results in Connection Functions

When a Connection Function is configured on the same distribution, CloudFront invokes it after OCSP validation completes. The connection object contains the OCSP status for the leaf and intermediate certificates:

```
{
  "clientCertificate": {
    "certificates": {
      "leaf": {
        "subject": "CN=client.example.com, O=Example Org",
        "issuer": "CN=Intermediate CA, O=Example Org",
        "serialNumber": "00:a7:30:9e:73:7b:3e:63:bd:b7:c0:7e:bf:d5:c9:86",
        "validity": {
          "notBefore": "2024-01-01T00:00:00Z",
          "notAfter": "2025-01-01T00:00:00Z"
        },
        "sha256Fingerprint":
 "AB:CD:EF:12:34:56:78:90:AB:CD:EF:12:34:56:78:90:AB:CD:EF:12:34:56:78:90:AB:CD:EF:12:34:56:78:
        "ocspEndpoint": "http://ocsp.example.org"
      },
      "intermediates": [
        {
          "subject": "CN=Intermediate CA, O=Example Org",
          "issuer": "CN=Root CA, O=Example Org",
          "serialNumber": "00:a7:30:9e:73:7b:3e:63:bd:b7:c0:7e:bf:d5:c9:86",
          "validity": {
            "notBefore": "2020-01-01T00:00:00Z",
            "notAfter": "2030-01-01T00:00:00Z"
          },
          "sha256Fingerprint":
 "12:34:56:78:90:AB:CD:EF:12:34:56:78:90:AB:CD:EF:12:34:56:78:90:AB:CD:EF:12:34:56:78:90:AB:CD:
          "ocspEndpoint": "http://ocsp.example.org"
        }
      ]
    },
    "revocationStatus": {
      "chainValidity": "Valid",           // "Valid" | "Invalid" | "Unknown"
      "certificates": {
```

```
        "leaf": {
          "method": "OCSP",               // "OCSP"
          "status": "Good",               // "Good" | "Revoked" | "Unknown" | "Error"
          "serialNumber": "00:a7:30:9e:73:7b:3e:63:bd:b7:c0:7e:bf:d5:c9:86"
        },
        "intermediates": [
          {
            "method": "OCSP",             // "OCSP"
            "status": "Error",            // "Good" | "Revoked" | "Unknown" | "Error"
            "errorType": "InternalError", // "InternalError" | "OCSP response
 verification failed: {Type}"
            "serialNumber": "00:a7:30:9e:73:7b:3e:63:bd:b7:c0:7e:bf:d5:c9:86"
          }
        ]
      }
    }
  },
  "clientIp":"127.0.0.1",
  "endpoint":"d123.cloudfront.net",
  "distributionId":"E1NXS4MQZH501R",
  "connectionId":"xdzQ6lJUDUt8b7OuqOD8lmzOC9HcMaXPmhH5ZdzLCZpKxqzfCPpR4A=="
}
```

The `chainValidity` field can be `Valid`, `Invalid`, or `Unknown`. Individual certificate `status` values can be `Good`, `Revoked`, `Unknown`, or `Error`. The `errorType` field contains `InternalError` or `OCSP response verification failed: {Type}` when the status is `Error`.

The Connection Function can override the OCSP result — for example, allowing a revoked certificate from a trusted IP range, or denying a certificate that OCSP reports as "good" based on additional business logic.

> **Note**
>
> The `errorType` field will only be present if the status is `Error`.

Example: Custom OCSP handling with trusted IP exception

```
function connectionHandler(connection) {
    var revocationStatus = connection.clientCertificate.revocationStatus;
```

```
    var trustedIP = (connection.clientIp === "[IP_ADDRESS]");

    if (revocationStatus.chainValidity === "Invalid") {
        if (trustedIP) {
            connection.allow();
        } else {
            connection.deny();
        }
    } else if (revocationStatus.certificates.leaf.status === "Error") {
        console.log(revocationStatus.certificates.leaf.errorType);
        connection.deny();
    } else {
        connection.allow();
    }
}
```

OCSP failure behavior

When CloudFront cannot determine a certificate's revocation status — because the OCSP responder is unreachable, returns an error, or returns "unknown" — CloudFront denies the connection by default. To implement soft-fail behavior, use a Connection Function. Connection Functions will not be executed if OCSP is enabled and the OCSP URL is absent in the client certificates. The OCSP result is available in the connection object, and your function can allow connections when the OCSP status is undetermined:

```
async function connectionHandler(connection) {
    var revocationStatus = connection.clientCertificate.revocationStatus;

    if (revocationStatus.certificates.leaf.status === "Error" ||
        revocationStatus.certificates.leaf.status === "Unknown") {
        // OCSP responder unreachable — allow connection (soft-fail)
        connection.logCustomData(`OCSP_SOFT_FAIL:
${revocationStatus.certificates.leaf.errorType}`);
        return connection.allow();
    }

    if (revocationStatus.chainValidity === "Invalid") {
        return connection.deny();
    }

    return connection.allow();
}
```

Certificate revocation with CloudFront Functions and KeyValueStore

You can use CloudFront Connection Functions with KeyValueStore to implement certificate revocation checking without any external dependency. You maintain a list of revoked certificate serial numbers in a KeyValueStore, and your Connection Function checks each client certificate against this list during the TLS handshake.

The certificate revocation process works as follows:

1. Store revoked certificate serial numbers in a CloudFront KeyValueStore.
2. When a client presents a certificate, your Connection Function is invoked.
3. The function checks the certificate's serial number against the KeyValueStore.
4. If the serial number is found in the store, the certificate is revoked.
5. Your function denies the connection for revoked certificates.

This approach provides near-real-time revocation checking across CloudFront global edge network.

To implement this approach, you need:

- A distribution configured with viewer mTLS
- A KeyValueStore containing revoked certificate serial numbers
- A Connection Function that queries the KeyValueStore to check certificate status

When a client connects, CloudFront validates the certificate against the trust store, then runs your Connection Function. Your function checks the certificate serial number against the KeyValueStore and allows or denies the connection.

Step 1: Create a KeyValueStore for revoked certificates

Prepare your revoked certificate serial numbers in JSON format:

```
{
  "data": [
    { "key": "ABC123DEF456", "value": "" },
    { "key": "789XYZ012GHI", "value": "" }
  ]
}
```

Upload this JSON file to an S3 bucket, then create the KeyValueStore:

```
aws s3 cp revoked-serials.json s3://your-bucket-name/revoked-serials.json

aws cloudfront create-key-value-store \
  --name revoked-serials-kvs \
  --import-source '{
    "SourceType": "S3",
    "SourceARN": "arn:aws:s3:::your-bucket-name/revoked-serials.json"
  }'
```

Wait for the KeyValueStore to finish provisioning. Check the status with:

```
aws cloudfront get-key-value-store --name "revoked-serials-kvs"
```

Step 2: Create the revocation Connection Function

Create a Connection Function that checks certificate serial numbers against the KeyValueStore:

```
aws cloudfront create-connection-function \
  --name "revocation-control" \
  --connection-function-config file://connection-function-config.json \
  --connection-function-code file://connection-function-code.txt
```

The configuration file specifies the KeyValueStore association:

```
{
  "Runtime": "cloudfront-js-2.0",
  "Comment": "A function that implements revocation control via KVS",
  "KeyValueStoreAssociations": {
    "Quantity": 1,
    "Items": [
      {
        "KeyValueStoreArn": "arn:aws:cloudfront::account-id:key-value-store/kvs-id"
      }
    ]
  }
}
```

Sample Connection Function code:

```
import cf from 'cloudfront';

async function connectionHandler(connection) {
```

```
    const kvsHandle = cf.kvs();

    // Get client serial number from client certificate
    const clientSerialNumber =
        connection.clientCertificate.certificates.leaf.serialNumber;

    // Check KVS to see if serial number exists as a key
    // Remove : from the clientSerialNumber if KVS entries don't have it
    const serialNumberExistsInKvs =
        await kvsHandle.exists(clientSerialNumber.replaceAll(":", ""));

    // Deny connection if serial number exists in KVS
    if (serialNumberExistsInKvs) {
        console.log("Connection denied – certificate revoked");
        connection.logCustomData("Connection denied – certificate revoked");
        return connection.deny();
    }

    // Allow connections that don't exist in KVS
    console.log("Connection allowed");
    return connection.allow();
}
```

Step 3: Test your revocation function

Use the CloudFront console to test your Connection Function with sample certificates. Navigate to the Connection Function in the console and use the Test tab.

- Paste a sample certificate in PEM format into the test interface.
- Optionally specify a client IP address for testing IP-based logic.
- Choose **Test function** to see the execution results.
- Review the execution logs to verify your function logic.

Test with both valid and revoked certificates to make sure your function handles both scenarios correctly.

Step 4: Associate the function to your distribution

After you publish your Connection Function, associate it with your mTLS-enabled distribution to activate certificate revocation checking. Navigate to your distribution settings, scroll to the "Viewer mutual authentication (mTLS)" section, select your Connection Function, and save the changes.

Advanced revocation strategies

Combine OCSP with Connection Function logic

You can enable OCSP for CA-authoritative revocation checking and layer a Connection Function on top for custom policies. The Connection Function receives the OCSP result and can apply additional logic:

- **Grace periods** — Allow revoked certificates from internal networks for a defined period during certificate rotation.
- **Emergency access** — Permit connections from specific IPs even when OCSP reports revoked status.
- **Custom deny logic** — Block certificates that OCSP reports as "good" based on your own revocation data in KeyValueStore.

Example OCSP + KVS combined revocation

```
import cf from 'cloudfront';

async function connectionHandler(connection) {
    var kvsHandle = cf.kvs();
    var revocationStatus = connection.clientCertificate.revocationStatus;
    var serialNumber = connection.clientCertificate.certificates.leaf.serialNumber;

    // Check your own revocation list first (immediate revocation, no cache delay)
    var inKvs = await kvsHandle.exists(serialNumber.replaceAll(":", ""));
    if (inKvs) {
        connection.logCustomData("KVS_REVOKED:" + serialNumber);
        return connection.deny();
    }

    // Then check OCSP result
    if (revocationStatus.chainValidity === 'Valid' &&
 revocationStatus.certificates.leaf.status === "Revoked") {
        // OCSP says revoked — allow grace period from trusted IPs
        if (connection.clientIp.startsWith("10.0.")) {
            connection.logCustomData("GRACE_PERIOD:" + serialNumber + ":" +
 connection.clientIp);
            return connection.allow();
        }
        connection.logCustomData("OCSP_REVOKED:" + serialNumber);
```

```
        return connection.deny();
    }

    connection.allow();
}
```

This pattern gives you immediate revocation by using KVS (no cache delay) plus CA-authoritative revocation by using OCSP, with custom exception handling in between.

Helper methods for mutual TLS

Use the `cf.mtls` helper functions in viewer request functions to rename, reformat, or combine client certificate headers before forwarding the request to your origin. For details, see the section called “Helper methods for mutual TLS”.

Additional validation modes

Required mode is the default for CloudFront mutual TLS — CloudFront validates every client certificate and denies connections that fail. However, to support additional use cases — such as applications that serve a mix of mTLS-authenticated and unauthenticated clients, or environments where the origin server performs its own mTLS validation — CloudFront supports two additional modes.

Optional mode is designed for applications with mixed client populations. For example, a portal that serves public content to unauthenticated browsers while restricting API endpoints to clients presenting valid certificates. Or a migration scenario where you are gradually onboarding clients to mTLS and need to allow both authenticated and unauthenticated connections during the transition.

Passthrough mode is designed for customers whose origin servers already perform mTLS validation. For example, services migrating from on-premises reverse proxies, other CDNs, or Application Load Balancers that already handle certificate verification. Passthrough mode allows these customers to place CloudFront in front of their applications without re-implementing validation logic at the edge.

Optional mode

Client certificate validation Optional mode

CloudFront offers an alternative Optional client certificate validation mode that validates client certificates that are presented but allows access to clients that don't present certificates.

Optional mode behavior

- Grants connection to clients with valid certificates (invalid certificates are denied).
- Allows connection to clients without certificates.
- Allows mixed client authentication scenarios through a single distribution.

Optional mode is ideal for gradual migration to mTLS authentication, supporting clients with certificates and clients without certificates, or maintaining backward compatibility with legacy clients.

> **Note**
>
> In optional mode, Connection Functions are still invoked even when clients don't present certificates. With this, you can implement custom logic such as logging client IP addresses or applying different policies based on whether certificates are presented.

To configure optional mode (Console)

1. In your distribution settings, navigate to the **General** tab, choose **Edit**.
2. Scroll to the **Viewer mutual authentication (mTLS)** section within the **Connectivity** container.
3. For **Client certificate validation mode**, select **Optional**.
4. Save changes.

To configure optional mode (AWS CLI)

The following example shows how to configure optional mode:

```
"ViewerMtlsConfig": {
    "Mode": "optional",
    ...other settings
}
```

Customize certificate headers

Use the mTLS helper functions in viewer-request CloudFront Functions to rename, reformat, or combine the certificate headers before they reach your origin. This is useful when migrating from other services that use different header names or certificate encoding formats.

Passthrough mode

With passthrough mode, you can use CloudFront with your existing mTLS implementations at your origins. CloudFront terminates the TLS connection and forwards the client certificate to your origin as HTTP headers. Your origin performs all certificate validation — including chain verification, revocation checking, and custom policy enforcement.

How passthrough mode works

1. The client connects to CloudFront and presents a client certificate during the TLS handshake.
2. CloudFront completes the TLS handshake without validating the certificate against a trust store.
3. CloudFront adds the client certificate and certificate chain as HTTP headers on the request.
4. The request is forwarded to your origin. No content is cached.
5. Your origin validates the certificate and processes the request.

Clients can also connect without presenting a certificate. Your origin or Connection Function handles empty certificate scenarios.

> **Note**
>
> In passthrough mode, Connection Functions are still invoked even when clients don't present certificates. With this, you can implement custom logic such as logging client IP addresses or applying different policies based on whether certificates are presented.

Configuration requirements

- **No trust store** — The distribution must have zero trust store associations.
- **Caching disabled** — All cache behaviors must use the managed `CachingDisabled` cache policy.
- **Origin Shield prohibited** — Origin Shield cannot be enabled.
- **Lambda@Edge prohibited** — Lambda@Edge function associations are not allowed.
- **Origin request policy required** — You must allowlist the `Client-Cert` and `Client-Cert-Chain` headers in your origin request policy for the origin to receive them.
- **Certificate-chain-depth** — CloudFront allows a maximum depth of 4 for client certificate chain to be forwarded to the origins.

Enable passthrough mode

Console

1. Update all cache behaviors to use the managed `CachingDisabled` cache policy.
2. Open the CloudFront console and choose your distribution.
3. Choose the **General** tab, then choose **Edit** under Settings.
4. Under **Viewer mutual authentication (mTLS)**, select **Passthrough**.
5. Choose **Save changes**.

AWS CLI

Make sure all cache behaviors reference the managed `CachingDisabled` policy, then update the distribution configuration:

```
{
  "ViewerMtlsConfig": {
    "Mode": "passthrough"
  }
}
```

Headers forwarded to origin

CloudFront adds the following headers to the request sent to your origin:

- `Client-Cert` — The end-entity (leaf) certificate presented by the client, base64-encoded.
- `Client-Cert-Chain` — The certificate chain (excluding the leaf), as a structured field list. Each certificate is base64-encoded. `Client-Cert-Chain` is a List type header. It might appear multiple times in a request. Concatenating all values preserves the original chain order. `Client-Cert-Chain` is omitted when the client presents only a single certificate.

CloudFront drops any incoming `Client-Cert` or `Client-Cert-Chain` headers from the client request before adding the actual certificate data. This prevents header spoofing.

Customize certificate headers

Use the mTLS helper functions in viewer-request CloudFront Functions to rename, reformat, or combine the certificate headers before they reach your origin. This is useful when migrating from other services that use different header names or certificate encoding formats.

Connection logs for mutual TLS

CloudFront can log each mutual TLS connection, including the client certificate and the reason that a connection failed. For details, see Connection logs.

Origin mutual TLS with CloudFront

Mutual TLS Authentication (Mutual Transport Layer Security Authentication — mTLS) is a security protocol that extends standard TLS authentication by requiring bidirectional certificate-based authentication, where both client and server must prove their identity before establishing a secure connection.

Viewer mTLS vs Origin mTLS

Mutual authentication (mTLS) can be enabled between viewers and your CloudFront distribution (viewer mTLS) and/or also between your CloudFront distribution and the origin (origin mTLS). This documentation pertains to origin mTLS configuration. For viewer mTLS configuration refer to: Mutual TLS authentication with CloudFront (Viewer mTLS).

Origin mTLS enables CloudFront to authenticate itself to your origin servers using client certificates. With origin mTLS, you can make sure that only your authorized CloudFront distributions can establish connections with your application servers, helping protect against unauthorized access attempts.

> **Note**
>
> In origin mTLS connections, CloudFront acts as the client and presents its client certificate to your origin server during the TLS handshake. CloudFront does not perform validation of the client certificate's validity or revocation status—this is the responsibility of your origin server. Your origin infrastructure must be configured to validate the client certificate against its trust store, check certificate expiration, and perform revocation checks (such as CRL or OCSP validation) according to your security requirements. CloudFront role is limited to presenting the certificate; all certificate validation logic and security policies are enforced by your origin servers.

How it works

In a standard TLS handshake between CloudFront and an origin, only the origin server presents a certificate to prove its identity to CloudFront. With origin mTLS, the authentication process becomes bidirectional. When CloudFront attempts to connect to your origin server, CloudFront presents a client certificate during the TLS handshake. Your origin server validates this certificate against its trust store before establishing the secure connection.

Use cases

Origin mTLS addresses several critical security scenarios where traditional authentication methods create operational overhead:

- **Hybrid and multi-cloud security** - You can secure connections between CloudFront and origins hosted outside AWS or public origins on AWS. This eliminates the need to manage IP allowlists or custom header solutions, providing consistent certificate-based authentication across AWS, on-premises data centers, and third-party providers. Media companies, retailers, and enterprises operating distributed infrastructure benefit from standardized security controls across their entire infrastructure.
- **B2B API and backend security** - You can protect your backend APIs and microservices from direct access attempts while maintaining CloudFront performance benefits. SaaS platforms, payment processing systems, and enterprise applications with strict authentication requirements can verify that API requests originate only from authorized CloudFront distributions, preventing man-in-the-middle attacks and unauthorized access attempts.

Important: Origin Server Requirements

Origin mTLS requires your origin servers to be configured to support mutual TLS authentication. Your origin infrastructure must be capable of:

- Requesting and validating client certificates during TLS handshakes
- Maintaining a trust store with the Certificate Authority certificates that issued CloudFront client certificates
- Logging and monitoring mutual TLS connection events
- Managing certificate validation policies and handling authentication failures

CloudFront handles the client-side certificate presentation, but your origin servers are responsible for validating these certificates and managing the mutual TLS connection. Make sure your origin infrastructure is properly configured before enabling origin mTLS in CloudFront.

Getting started

To implement origin mTLS with CloudFront, you'll need to import the client certificate in AWS Certificate Manager, configure your origin server to require mutual TLS, and enable origin mTLS on your CloudFront distribution. The following sections provide step-by-step instructions for each configuration task.

Topics

- Certificate management with AWS Certificate Manager
- Enable origin mutual TLS for CloudFront distributions
- Using CloudFront Functions with origin mutual TLS

Certificate management with AWS Certificate Manager

AWS Certificate Manager (ACM) stores the client certificates that CloudFront presents to your origin servers during origin mutual TLS authentication.

Certificate Authority support

CloudFront origin mTLS requires client certificates with Extended Key Usage (EKU) for TLS Client Authentication. Due to this requirement, you must issue certificates from your Certificate Authority and import them into AWS Certificate Manager. ACM automatic certificate provisioning and renewal features are not available for origin mTLS client certificates. CloudFront origin mTLS supports client certificates from two sources:

- **AWS Private Certificate Authority:** You can issue certificates from AWS Private CA using certificate templates that include TLS Client Authentication in the Extended Key Usage field (such as the EndEntityClientAuthCertificate template). After issuing the certificate from AWS Private CA, you must import it into ACM in the US East (N. Virginia) Region (us-east-1). This approach provides the security benefits of AWS Private CA while giving you control over certificate lifecycle management.
- **Third-party private Certificate Authorities:** You can also issue certificates from your existing private Certificate Authority infrastructure and import them into ACM. This allows you to maintain your current certificate management processes while leveraging CloudFront origin

mTLS capabilities. Certificates must include TLS Client Authentication in the Extended Key Usage field and must be in PEM format with the certificate, private key, and certificate chain.

> **Important**
>
> For both AWS Private CA and third-party CAs, you are responsible for monitoring certificate expiration dates and importing renewed certificates into ACM before expiration. ACM automatic renewal feature does not apply to imported certificates used for origin mTLS.

Certificate requirements and specifications

Client certificate requirements

- **Format:** PEM (Privacy Enhanced Mail) format
- **Components:** Certificate, private key, and certificate chain
- **Maximum certificate chain depth:** 3 (leaf certificate + intermediate certificate + root certificate)
- **Maximum certificate chain size:** 64 KB
- **Certificate size:** Cannot exceed 96 KB
- **Maximum private key size:** 5 KB (ACM restriction)
- **Maximum unique origin mTLS certificate ARNs that can be added or modified per CloudFront distribution create or update API call:** 5
- **Region:** Certificates must be stored in ACM in the US East (N. Virginia) Region (us-east-1)

Supported certificate specifications

- **Certificate type:** X.509v3
- **Public key algorithms:**
  - RSA: 2048-bit
  - ECDSA: P-256
- **Signature algorithms:**
  - SHA256, SHA384, SHA512 with RSA
  - SHA256, SHA384, SHA512 with ECDSA
  - SHA256, SHA384, SHA512 with RSASSA-PSS with MGF1

- **Extended Key Usage (required):** The certificate requires the Extended Key Usage (EKU) extension set to TLS Client Authentication, making sure it is authorized for mTLS purposes

Server certificate requirements

Your origin servers must present certificates from publicly trusted Certificate Authorities during the mutual TLS handshake. For complete details on origin server certificate requirements, see Requirements for using SSL/TLS certificates with CloudFront.

Request or import a certificate

Before enabling origin mTLS, you must have a client certificate available in ACM.

Request and import a certificate from AWS Private CA

Prerequisites:

- An AWS Private Certificate Authority configured in your account
- Permission to issue certificates from AWS Private CA
- Permission to import certificates into ACM
- A certificate template ARN with `Extended key usage:TLS web client authentication` which suits your use-case
- Install OpenSSL, AWS CLI, and jq (for parsing JSON).

To request a certificate from PCA and import into ACM (AWS CLI)

1. Set your Private CA ARN in a variable for easier reuse.

```
PCA_ARN="arn:aws:acm-pca:region:account:certificate-authority/12345678..."
```

2. Use OpenSSL to generate an ECDSA P-256 private key (prime256v1 curve) and Certificate Signing Request (CSR), making sure the -nodes flag is used to keep the private key unencrypted as required for ACM import.

```
openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -nodes \
    -keyout private.key \
    -out request.csr \
    -subj "/CN=client.example.com"
```

3. Submit the CSR to your AWS Private CA to issue a certificate, which returns the ARN of the newly issued certificate.

```
CERT_ARN=$(aws acm-pca issue-certificate \
    --certificate-authority-arn "$PCA_ARN" \
    --csr fileb://request.csr \
    --signing-algorithm "SHA256WITHECDSA" \
    --validity Value=365,Type="DAYS" \
    --template-arn arn:aws:acm-pca:::template/EndEntityCertificate/V1 \
    --query 'CertificateArn' --output text)
```

4. Retrieve the certificate bundle from AWS PCA using the get-certificate command, which returns both the leaf certificate and chain, then use jq to separate them into distinct files.

```
# Retrieve the full certificate bundle in JSON format
aws acm-pca get-certificate \
    --certificate-authority-arn "$PCA_ARN" \
    --certificate-arn "$CERT_ARN" \
    --output json > full_cert.json

# Split into Leaf and Chain
jq -r '.Certificate' full_cert.json > leaf_cert.pem
jq -r '.CertificateChain' full_cert.json > cert_chain.pem
```

5. Import the unencrypted private key, leaf certificate, and certificate chain into AWS ACM, using the fileb:// protocol to correctly handle binary file data in the CLI.

```
aws acm import-certificate \
    --certificate fileb://leaf_cert.pem \
    --private-key fileb://private.key \
    --certificate-chain fileb://cert_chain.pem \
    --region us-east-1 \
    --query 'CertificateArn' \
    --output text
```

Import a certificate from a third-party CA

Prerequisites:

- A certificate, unencrypted private key, and certificate chain from your Certificate Authority in PEM format

- The certificate must include Extended Key Usage for TLS Client Authentication
- Permissions to import certificates into ACM

To import a certificate into ACM (AWS CLI)

```
aws acm import-certificate \
  --certificate fileb://certificate.pem \
  --private-key fileb://private-key.pem \
  --certificate-chain fileb://certificate-chain.pem \
  --region us-east-1 \
  --query 'CertificateArn' \
  --output text
```

Next steps

After obtaining or importing your client certificate in ACM, you can configure your origin server to require mutual TLS authentication and enable origin mTLS on your CloudFront distribution. For instructions on enabling origin mTLS in CloudFront, see the next section "Enable origin mutual TLS for CloudFront distributions."

Enable origin mutual TLS for CloudFront distributions

After obtaining a client certificate through AWS Certificate Manager and configuring your origin server to require mutual TLS (mTLS), you can enable origin mTLS on your CloudFront distribution.

Prerequisites and requirements

Before enabling origin mTLS on a CloudFront distribution, make sure you have:

- A client certificate stored in AWS Certificate Manager in the US East (N. Virginia) Region (us-east-1)
- Origin servers configured to require mutual TLS authentication and validate client certificates
- Origin servers presenting certificates from publicly trusted Certificate Authorities
- Permissions to modify CloudFront distributions
- Origin mTLS is only available on Business, Premium plans or Pay as you go pricing plans.

Note

Origin mTLS can be configured for custom origins (including origins hosted outside AWS) and AWS origins that support mutual TLS such as Application Load Balancer and API Gateway.

Important

You can't use origin mTLS with the following CloudFront features:

- **Embedded points of presence (POPs)** – CloudFront doesn't support origin mTLS on Embedded POPs.
- **Origin request and origin response triggers with Lambda@Edge** – You can't use a Lambda@Edge function in an origin request or origin response trigger with origin mTLS. For more information, see Add triggers for a Lambda@Edge function.
- **VPC origins** – You can't use origin mTLS with VPC origins. For more information, see Restrict access with VPC origins.

Enable origin mTLS

Per-origin configuration allows you to specify different client certificates for different origins within the same distribution. This approach provides maximum flexibility when your origins have different authentication requirements.

For new distributions (Console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **Create distribution**
3. Select a pricing plan: Choose either **Business** or **Premium** or **Pay As You Go** (Origin mTLS is not available on the Free plan)
4. In the Origin settings section, choose Origin Type as Other
5. In the **Origin settings** section, choose **Customize origin settings**
6. Configure your first origin (domain name, protocol, and so on)

7. In the origin configuration, find **mTLS**
8. Toggle **mTLS** to On
9. For **Client certificate**, select your certificate from AWS Certificate Manager
10. (Optional) Add additional origins with their own origin mTLS configurations
11. Complete the remaining distribution settings and choose **Create distribution**

For existing distributions (Console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. From the distribution list, select the distribution you want to modify. (Note: Make sure your distribution is on a **Pro or Premium or Pay As You Go** pricing plan. If not, you must upgrade your pricing plan before enabling origin mTLS)
3. Choose the **Origins** tab
4. Select the origin you want to configure and choose **Edit**
5. In the origin settings, find **mTLS**
6. Toggle **mTLS** to On
7. For **Client certificate**, select your certificate from AWS Certificate Manager. (Note: Only client certificates with the EKU (Extended Key Usage) property set to "TLS Client Authentication" will be listed)
8. Choose **Save changes**
9. Repeat for additional origins as needed

Using AWS CLI

For per-origin configuration, specify the origin mTLS settings within each origin's configuration:

```
{
  "Origins": {
    "Quantity": 2,
    "Items": [
      {
        "Id": "origin-1",
        "DomainName": "api.example.com",
        "CustomOriginConfig": {
          "HTTPSPort": 443,
```

```
          "OriginProtocolPolicy": "https-only"
        },
        "OriginMtlsConfig": {
          "ClientCertificateArn": "arn:aws:acm:us-east-1:123456789012:certificate/
cert-1"
        }
      },
      {
        "Id": "origin-2",
        "DomainName": "backend.example.com",
        "CustomOriginConfig": {
          "HTTPSPort": 443,
          "OriginProtocolPolicy": "https-only"
        },
        "OriginMtlsConfig": {
          "CertificateArn": "arn:aws:acm:us-east-1:123456789012:certificate/cert-2"
        }
      }
    ]
  }
}
```

Note

CloudFront will not provide the client certificate if the server does not request it, allowing the connection to proceed normally.

Next steps

After enabling origin mTLS on your CloudFront distribution, you can monitor authentication events using CloudFront access logs.

Using CloudFront Functions with origin mutual TLS

CloudFront Functions provides lightweight, serverless compute at the edge to customize content delivery. When using origin mutual TLS with CloudFront Functions, there are specific behaviors and limitations to be aware of regarding origin selection and manipulation.

Supported CloudFront Functions operations

CloudFront Functions can interact with mTLS-enabled origins in the following ways:

updateRequestOrigin()

The updateRequestOrigin() function supports limited modifications when working with mTLS-enabled origins:

- **Switching between origin mTLS origins:** You can update the request to route to a different origin that uses origin mTLS, provided both origins use the **same client certificate**. This allows you to implement custom routing logic while maintaining mutual TLS authentication. Switching between origins that make use of different certificates is supported through the `selectRequestOriginById()` and `createRequestOriginGroup()` APIs.
- **Disabling origin mTLS:** You can switch from an mTLS-enabled origin to a non-mTLS origin by setting `mTLSConfig: 'off'` in the function. This provides flexibility to conditionally disable mutual TLS authentication based on request characteristics.

Example: Switching between origin mTLS origins with the same certificate

```
import cf from 'cloudfront';

function handler(event) {
    var request = event.request;

    // Route to different origin based on request path
    if (request.uri.startsWith('/api/v2')) {
        cf.updateRequestOrigin({
            "domainName": "api-v2.example.com",
            "mTLSConfig": "inherit",
            // If no value is provided for mTLSConfig, it defaults to inherit
            // Both origins must use the same certificate
        });
    }

    return request;
}
```

Example: Conditionally disabling origin mTLS

```
import cf from 'cloudfront';

function handler(event) {
    var request = event.request;
```

```
    // Disable mTLS for specific paths
    if (request.uri.startsWith('/public')) {
        cf.updateRequestOrigin({
            "domainName": "public-origin.example.com",
            "mTLSConfig": "off"
        });
    }

    return request;
}
```

selectRequestOriginById()

The `selectRequestOriginById()` function supports selecting origins that have mutual TLS (origin) enabled. You can use this function to dynamically route requests to mTLS-enabled origins configured in your distribution. When selecting a mutual TLS (origin) enabled origin by ID, CloudFront uses the client certificate configured for that origin in the distribution settings.

Example: Selecting a mutual TLS (origin) enabled origin by ID

```
import cf from 'cloudfront';

function handler(event) {
    var request = event.request;

    // Select mTLS-enabled origin based on request characteristics
    if (request.uri.startsWith('/secure-api')) {
        cf.selectRequestOriginById("mtls-origin-1");
    }

    return request;
}
```

createRequestOriginGroup()

The `createRequestOriginGroup()` function supports creating origin groups that include mutual TLS (origin) enabled origins. You can dynamically create origin groups with mTLS-enabled origins for failover scenarios within CloudFront Functions.

Example: Creating an origin group with mutual TLS (origin) enabled origins

```
import cf from 'cloudfront';
```

```
function handler(event) {
    // Create origin group with mTLS-enabled primary and failover origins
    cf.createRequestOriginGroup({
        "originIds": ["mtls-origin-primary", "mtls-origin-failover"],
        "failoverCriteria": {
            "statusCodes": [500, 502, 503, 504]
        }
    });

    return event.request;
}
```

Serve private content with signed URLs and signed cookies

Many companies that distribute content over the internet want to restrict access to documents, business data, media streams, or content that is intended for selected users, for example, users who have paid a fee. To securely serve this private content by using CloudFront, you can do the following:

- Require that your users access your private content by using special CloudFront signed URLs or signed cookies.
- Require that your users access your content by using CloudFront URLs, not URLs that access content directly on the origin server (for example, Amazon S3 or a private HTTP server). Requiring CloudFront URLs isn't necessary, but we recommend it to prevent users from bypassing the restrictions that you specify in signed URLs or signed cookies.

For more information, see Restrict access to files.

How to serve private content

To configure CloudFront to serve private content, do the following tasks:

1. (Optional but recommended) Require your users to access your content only through CloudFront. The method that you use depends on whether you're using Amazon S3 or custom origins:
   - **Amazon S3** – See the section called “Restrict access to an Amazon S3 origin”.
   - **Custom origin** – See Restrict access to files on custom origins.

Custom origins include Amazon EC2, Amazon S3 buckets configured as website endpoints, Elastic Load Balancing, and your own HTTP web servers.

2. Specify the *trusted key groups* or *trusted signers* that you want to use to create signed URLs or signed cookies. We recommend that you use trusted key groups. For more information, see Specify signers that can create signed URLs and signed cookies.
3. Write your application to respond to requests from authorized users either with signed URLs or with `Set-Cookie` headers that set signed cookies. Follow the steps in one of the following topics:

   - Use signed URLs
   - Use signed cookies

   If you're not sure which method to use, see Decide to use signed URLs or signed cookies.

Topics

- Restrict access to files
- Specify signers that can create signed URLs and signed cookies
- Decide to use signed URLs or signed cookies
- Use signed URLs
- Use signed cookies
- Linux commands and OpenSSL for base64 encoding and encryption
- Code examples for creating a signature for a signed URL

Restrict access to files

You can control user access to your private content in two ways:

- Restrict access to files in CloudFront caches.
- Restrict access to files in your origin by doing one of the following:
  - Set up an origin access control (OAC) for your Amazon S3 bucket.
  - Configure custom headers for a private HTTP server (a custom origin).

Restrict access to files in CloudFront caches

You can configure CloudFront to require that users access your files using either *signed URLs* or *signed cookies*. You then develop your application either to create and distribute signed URLs to authenticated users or to send `Set-Cookie` headers that set signed cookies for authenticated users. (To give a few users long-term access to a small number of files, you can also create signed URLs manually.)

When you create signed URLs or signed cookies to control access to your files, you can specify the following restrictions:

- An ending date and time, after which the URL is no longer valid.
- (Optional) The date and time that the URL becomes valid.
- (Optional) The IP address or range of addresses of the computers that can be used to access your content.

One part of a signed URL or a signed cookie is hashed and signed using the private key from a public–private key pair. When someone uses a signed URL or signed cookie to access a file, CloudFront compares the signed and unsigned portions of the URL or cookie. If they don't match, CloudFront doesn't serve the file.

You must use either RSA 2048 or ECDSA 256 private keys for signing URLs or cookies.

Restrict access to files in Amazon S3 buckets

You can optionally secure the content in your Amazon S3 bucket so that users can access it through the specified CloudFront distribution but cannot access it directly by using Amazon S3 URLs. This prevents someone from bypassing CloudFront and using the Amazon S3 URL to get content that you want to restrict access to. This step isn't required to use signed URLs, but we recommend it.

To require that users access your content through CloudFront URLs, you do the following tasks:

- Give a CloudFront *origin access control* permission to read the files in the S3 bucket.
- Create the origin access control and associate it with your CloudFront distribution.
- Remove permission for anyone else to use Amazon S3 URLs to read the files.

For more information, see the section called “Restrict access to an Amazon S3 origin” or the section called “Restrict access to an Amazon S3 Multi-Region Access Point origin”.

Restrict access to files on custom origins

If you use a custom origin, you can optionally set up custom headers to restrict access. For CloudFront to get your files from a custom origin, the files must be accessible by CloudFront using a standard HTTP (or HTTPS) request. But by using custom headers, you can further restrict access to your content so that users can access it only through CloudFront, not directly. This step isn't required to use signed URLs, but we recommend it.

To require that users access content through CloudFront, change the following settings in your CloudFront distributions:

Origin Custom Headers

Configure CloudFront to forward custom headers to your origin. See Configure CloudFront to add custom headers to origin requests.

Viewer Protocol Policy

Configure your distribution to require viewers to use HTTPS to access CloudFront. See Viewer protocol policy.

Origin Protocol Policy

Configure your distribution to require CloudFront to use the same protocol as viewers to forward requests to the origin. See Protocol (custom origins only).

After you've made these changes, update your application on your custom origin to only accept requests that include the custom headers that you've configured CloudFront to send.

The combination of **Viewer Protocol Policy** and **Origin Protocol Policy** makes sure that the custom headers are encrypted in transit. However, we recommend that you periodically do the following to rotate the custom headers that CloudFront forwards to your origin:

1. Update your CloudFront distribution to begin forwarding a new header to your custom origin.
2. Update your application to accept the new header as confirmation that the request is coming from CloudFront.
3. When requests no longer include the header that you're replacing, update your application to no longer accept the old header as confirmation that the request is coming from CloudFront.

Specify signers that can create signed URLs and signed cookies

Topics

- Choose between trusted key groups (recommended) and AWS accounts
- Create key pairs for your signers
- Reformat the private key (.NET and Java only)
- Add a signer to a distribution
- Rotating key pairs

To create signed URLs or signed cookies, you need a *signer*. A signer is either a trusted key group that you create in CloudFront, or an AWS account that contains a CloudFront key pair. We recommend that you use trusted key groups with signed URLs and signed cookies. For more information, see Choose between trusted key groups (recommended) and AWS accounts.

The signer has two purposes:

- As soon as you add the signer to your distribution, CloudFront starts to require that viewers use signed URLs or signed cookies to access your files.
- When you create signed URLs or signed cookies, you use the private key from the signer's key pair to sign a portion of the URL or the cookie. When someone requests a restricted file, CloudFront compares the signature in the URL or cookie with the unsigned URL or cookie, to verify that it hasn't been tampered with. CloudFront also verifies that the URL or cookie is valid, meaning, for example, that the expiration date and time hasn't passed.

When you specify a signer, you also indirectly specify the files that require signed URLs or signed cookies by adding the signer to a cache behavior. If your distribution has only one cache behavior, viewers must use signed URLs or signed cookies to access any file in the distribution. If you create multiple cache behaviors and add signers to some cache behaviors and not to others, you can require that viewers use signed URLs or signed cookies to access some files and not others.

To specify the signers (the private keys) that are allowed to create signed URLs or signed cookies, and to add the signers to your CloudFront distribution, do the following tasks:

1. Decide whether to use a trusted key group or an AWS account as the signer. We recommend using a trusted key group. For more information, see Choose between trusted key groups (recommended) and AWS accounts.

2. For the signer that you chose in step 1, create a public–private key pair. For more information, see Create key pairs for your signers.
3. If you're using .NET or Java to create signed URLs or signed cookies, reformat the private key. For more information, see Reformat the private key (.NET and Java only).
4. In the distribution for which you're creating signed URLs or signed cookies, specify the signer. For more information, see Add a signer to a distribution.

Choose between trusted key groups (recommended) and AWS accounts

To use signed URLs or signed cookies, you need a *signer*. A signer is either a trusted key group that you create in CloudFront, or an AWS account that contains a CloudFront key pair. We recommend that you use trusted key groups, for the following reasons:

- With CloudFront key groups, you don't need to use the AWS account root user to manage the public keys for CloudFront signed URLs and signed cookies. AWS best practices recommend that you don't use the root user when you don't have to.
- With CloudFront key groups, you can manage public keys, key groups, and trusted signers using the CloudFront API. You can use the API to automate key creation and key rotation. When you use the AWS root user, you have to use the AWS Management Console to manage CloudFront key pairs, so you can't automate the process.
- Because you can manage key groups with the CloudFront API, you can also use AWS Identity and Access Management (IAM) permissions policies to limit what different users are allowed to do. For example, you can allow users to upload public keys, but not delete them. Or you can allow users to delete public keys, but only when certain conditions are met, such as using multi-factor authentication, sending the request from a particular network, or sending the request within a particular date and time range.
- With CloudFront key groups, you can associate a higher number of public keys with your CloudFront distribution, giving you more flexibility in how you use and manage the public keys. By default, you can associate up to four key groups with a single distribution, and you can have up to five public keys in a key group.

  When you use the AWS account root user to manage CloudFront key pairs, you can only have up to two active CloudFront key pairs per AWS account.

Create key pairs for your signers

Each signer that you use to create CloudFront signed URLs or signed cookies must have a public–private key pair. The signer uses its private key to sign the URL or cookies, and CloudFront uses the public key to verify the signature.

The way that you create a key pair depends on whether you use a trusted key group as the signer (recommended), or a CloudFront key pair. For more information, see the following sections. The key pair that you create must meet the following requirements:

- It must be either an SSH-2 RSA 2048 or ECDSA 256 key pair.
- It must be in base64-encoded PEM format.

To help secure your applications, we recommend that you rotate key pairs periodically. For more information, see Rotating key pairs.

Create a key pair for a trusted key group (recommended)

To create a key pair for a trusted key group, perform the following steps:

1. Create the public–private key pair.
2. Upload the public key to CloudFront.
3. Add the public key to a CloudFront key group.

For more information, see the following procedures.

To create a key pair

> **Note**
>
> The following steps use OpenSSL as an example of one method for creating a key pair. There are many other ways to create an RSA or ECDSA key pair.

1. Run one of the following example commands:

   - The following example command uses OpenSSL to generate an RSA key pair with a length of 2048 bits and save to the file named `private_key.pem`.

```
openssl genrsa -out private_key.pem 2048
```

- The following example command uses OpenSSL to generate an ECDSA key pair with the `prime256v1` curve and save to the file named `private_key.pem`.

```
openssl ecparam -name prime256v1 -genkey -noout -out privatekey.pem
```

2. The resulting file contains both the public and the private key. The following example command extracts the public key from the file named `private_key.pem`.

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

You upload the public key (in the `public_key.pem` file) later, in the following procedure.

To upload the public key to CloudFront

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation menu, choose **Public keys**.
3. Choose **Create public key**.
4. In the **Create public key** window, do the following:
   a. For **Key name**, type a name to identify the public key.
   b. For **Key value**, paste the public key. If you followed the steps in the preceding procedure, the public key is in the file named `public_key.pem`. To copy and paste the contents of the public key, you can:
      - Use the **cat** command on the macOS or Linux command line, like this:

```
cat public_key.pem
```

Copy the output of that command, then paste it into the **Key value** field.

- Open the `public_key.pem` file with a plaintext editor like Notepad (on Windows) or TextEdit (on macOS). Copy the contents of the file, then paste it into the **Key value** field.

c. (Optional) For **Comment**, add a comment to describe the public key.

When finished, choose **Add**.

5. Record the public key ID. You use it later when you create signed URLs or signed cookies, as the value of the `Key-Pair-Id` field.

To add the public key to a key group

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation menu, choose **Key groups**.
3. Choose **Add key group**.
4. On the **Create key group** page, do the following:

   a. For **Key group name**, type a name to identify the key group.

   b. (Optional) For **Comment**, type a comment to describe the key group.

   c. For **Public keys**, select the public key to add to the key group, then choose **Add**. Repeat this step for each public key that you want to add to the key group.

5. Choose **Create key group**.
6. Record the key group name. You use it later to associate the key group with a cache behavior in a CloudFront distribution. (In the CloudFront API, you use the key group ID to associate the key group with a cache behavior.)

Create a CloudFront key pair (not recommended, requires the AWS account root user)

> **Important**
>
> We recommend that you create a public key for a trusted key group instead of following these steps. For the recommended way to create public keys for signed URLs and signed cookies, see Create a key pair for a trusted key group (recommended).

You can create a CloudFront key pair in the following ways:

- Create a key pair in the AWS Management Console and download the private key. See the following procedure.
- Create an RSA key pair by using an application such as OpenSSL, and then upload the public key to the AWS Management Console. For more information about creating an RSA key pair, see Create a key pair for a trusted key group (recommended).

To create CloudFront key pairs in the AWS Management Console

1. Sign in to the AWS Management Console using the credentials of the AWS account root user.

   > **Important**
   >
   > IAM users can't create CloudFront key pairs. You must sign in using root user credentials to create key pairs.

2. Choose your account name, then choose **My Security Credentials**.
3. Choose **CloudFront key pairs**.
4. Confirm that you have no more than one active key pair. You can't create a key pair if you already have two active key pairs.
5. Choose **Create New Key Pair**.

   > **Note**
   >
   > You can also choose to create your own key pair and upload the public key. CloudFront key pairs support 1024, 2048, or 4096-bit keys.

6. In the **Create Key Pair** dialog box, choose **Download Private Key File**, and then save the file on your computer.

   > **Important**
   >
   > Save the private key for your CloudFront key pair in a secure location, and set permissions on the file so that only you can read it. If someone gets your private key, they can generate valid signed URLs and signed cookies and download your content.

You cannot get the private key again, so if you lose or delete it, you must create a new CloudFront key pair.

7. Record the key pair ID for your key pair. (In the AWS Management Console, this is called the **Access Key ID**.) You'll use it when you create signed URLs or signed cookies.

Reformat the private key (.NET and Java only)

If you're using .NET or Java to create signed URLs or signed cookies, you cannot use the private key from your key pair in the default PEM format to create the signature. Instead, do the following:

- **.NET framework** – Convert the private key to the XML format that the .NET framework uses. Several tools are available.
- **Java** – Convert the private key to DER format. One way to do this is with the following OpenSSL command. In the following command, `private_key.pem` is the name of the file that contains the PEM-formatted private key, and `private_key.der` is the name of the file that contains the DER-formatted private key after you run the command.

  ```
  openssl pkcs8 -topk8 -nocrypt -in private_key.pem -inform PEM -out private_key.der -
  outform DER
  ```

  To make sure that the encoder works correctly, add the JAR for the Bouncy Castle Java cryptography APIs to your project and then add the Bouncy Castle provider.

Add a signer to a distribution

A signer is the trusted key group (recommended) or CloudFront key pair that can create signed URLs and signed cookies for a distribution. To use signed URLs or signed cookies with a CloudFront distribution, you must specify a signer.

Signers are associated with cache behaviors. With this, you can require signed URLs or signed cookies for some files and not for others in the same distribution. A distribution requires signed URLs or cookies only for files that are associated with the corresponding cache behaviors.

Similarly, a signer can only sign URLs or cookies for files that are associated with the corresponding cache behaviors. For example, if you have one signer for one cache behavior and a different signer

for a different cache behavior, neither signer can create signed URLs or cookies for files that are associated with the other cache behavior.

Important

Before you add a signer to your distribution, do the following:

- Define the path patterns in cache behaviors and the sequence of cache behaviors carefully so you don't give users unintended access to your content or prevent them from accessing content that you want to be available to everyone.

  For example, suppose a request matches the path pattern for two cache behaviors. The first cache behavior does not require signed URLs or signed cookies and the second cache behavior does. Users will be able to access the files without using signed URLs or signed cookies because CloudFront processes the cache behavior that is associated with the first match.

  For more information about path patterns, see Path pattern.
- For a distribution that you're already using to distribute content, make sure you're ready to start generating signed URLs and signed cookies before you add a signer. When you add a signer, CloudFront rejects requests that don't include a valid signed URL or signed cookie.

You can add signers to your distribution using either the CloudFront console or the CloudFront API.

Console

The following steps show how to add a trusted key group as a signer. You can also add an AWS account as a trusted signer, but it's not recommended.

To add a signer to a distribution using the console

1. Record the key group ID of the key group that you want to use as a trusted signer. For more information, see Create a key pair for a trusted key group (recommended).
2. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
3. Choose the distribution whose files you want to protect with signed URLs or signed cookies.

> **Note**
>
> To add a signer to a new distribution, you specify the same settings that are described in step 6 when you create the distribution.

4. Choose the **Behaviors** tab.
5. Select the cache behavior whose path pattern matches the files that you want to protect with signed URLs or signed cookies, and then choose **Edit**.
6. On the **Edit Behavior** page, do the following:
   a. For **Restrict Viewer Access (Use Signed URLs or Signed Cookies)**, choose **Yes**.
   b. For **Trusted Key Groups or Trusted Signer**, choose **Trusted Key Groups**.
   c. For **Trusted Key Groups**, choose the key group to add, and then choose **Add**. Repeat if you want to add more than one key group.
7. Choose **Yes, Edit** to update the cache behavior.

API

You can use the CloudFront API to add a trusted key group as a signer. You can add a signer to an existing distribution or to a new distribution. In either case, specify the values in the `TrustedKeyGroups` element.

You can also add an AWS account as a trusted signer, but it's not recommended.

See the following topics in the *Amazon CloudFront API Reference*:

- **Update an existing distribution** – UpdateDistribution
- **Create a new distribution** – CreateDistribution

Rotating key pairs

We recommend that you periodically rotate (change) your key pairs for signed URLs and signed cookies. To rotate key pairs that you're using to create signed URLs or signed cookies without invalidating URLs or cookies that haven't expired yet, do the following tasks:

1. Create a new key pair, and add the public key to a key group. For more information, see Create a key pair for a trusted key group (recommended).

2. If you created a new key group in the previous step, add the key group to the distribution as a signer.

> **Important**
>
> Don't remove any existing public keys from the key group, or any key groups from the distribution yet. Only add the new ones.

3. Update your application to create signatures using the private key from the new key pair. Confirm that the signed URLs or cookies that are signed with the new private keys are working.
4. Wait until the expiration date has passed in URLs or cookies that were signed using the previous private key. Then remove the old public key from the key group. If you created a new key group in step 2, remove the old key group from your distribution.

Decide to use signed URLs or signed cookies

CloudFront signed URLs and signed cookies provide the same basic functionality: they allow you to control who can access your content. If you want to serve private content through CloudFront and you're trying to decide whether to use signed URLs or signed cookies, consider the following.

Use signed URLs in the following cases:

- You want to restrict access to individual files, for example, an installation download for your application.
- Your users are using a client (for example, a custom HTTP client) that doesn't support cookies.

Use signed cookies in the following cases:

- You want to provide access to multiple restricted files, for example, all of the files for a video in HLS format or all of the files in the subscribers' area of website.
- You don't want to change your current URLs.

If you are not currently using signed URLs, and if your (unsigned) URLs contain any of the following query string parameters, you cannot use either signed URLs or signed cookies:

- `Expires`
- `Policy`

- `Signature`
- `Key-Pair-Id`
- `Hash-Algorithm`

CloudFront assumes that URLs that contain any of those query string parameters are signed URLs, and therefore won't look at signed cookies.

Use both signed URLs and signed cookies

Signed URLs take precedence over signed cookies. If you use both signed URLs and signed cookies to control access to the same files and a viewer uses a signed URL to request a file, CloudFront determines whether to return the file to the viewer based only on the signed URL.

Use signed URLs

A signed URL includes additional information, for example, an expiration date and time, that gives you more control over access to your content. This additional information appears in a policy statement, which is based on either a canned policy or a custom policy. The differences between canned and custom policies are explained in the next two sections.

> **Note**
>
> You can create some signed URLs using canned policies and create some signed URLs using custom policies for the same distribution.

Topics

- Decide to use canned or custom policies for signed URLs
- How signed URLs work
- Decide how long signed URLs are valid
- When CloudFront checks expiration date and time in a signed URL
- Example code and third-party tools
- Create a signed URL using a canned policy
- Create a signed URL using a custom policy

Decide to use canned or custom policies for signed URLs

When you create a signed URL, you write a policy statement in JSON format that specifies the restrictions on the signed URL, for example, how long the URL is valid. You can use either a canned policy or a custom policy. Here's how canned and custom policies compare:

Description	Canned policy	Custom policy
You can reuse the policy statement for multiple files. To reuse the policy statement, you must use wildcard characters in the `Resource` object. For more information, see Values that you specify in the policy statement for a signed URL that uses a custom policy.)	No	Yes
You can specify the date and time that users can begin to access your content.	No	Yes (optional)
You can specify the date and time that users can no longer access your content.	Yes	Yes
You can specify the IP address or range of IP addresses of the users who can access your content.	No	Yes (optional)
The signed URL includes a base64-encoded version of the policy, which results in a longer URL.	No	Yes

For information about creating signed URLs using a *canned* policy, see Create a signed URL using a canned policy.

For information about creating signed URLs using a *custom* policy, see Create a signed URL using a custom policy.

How signed URLs work

Here's an overview of how you configure CloudFront and Amazon S3 for signed URLs and how CloudFront responds when a user uses a signed URL to request a file.

1. In your CloudFront distribution, specify one or more trusted key groups, which contain the public keys that CloudFront can use to verify the URL signature. You use the corresponding private keys to sign the URLs.

   CloudFront supports signed URLs with RSA 2048 and ECDSA 256 key signatures.

   For more information, see Specify signers that can create signed URLs and signed cookies.
2. Develop your application to determine whether a user should have access to your content and to create signed URLs for the files or parts of your application that you want to restrict access to. For more information, see the following topics:

   - Create a signed URL using a canned policy
   - Create a signed URL using a custom policy
3. A user requests a file for which you want to require signed URLs.
4. Your application verifies that the user is entitled to access the file: they've signed in, they've paid for access to the content, or they've met some other requirement for access.
5. Your application creates and returns a signed URL to the user.
6. The signed URL allows the user to download or stream the content.

   This step is automatic; the user usually doesn't have to do anything additional to access the content. For example, if a user is accessing your content in a web browser, your application returns the signed URL to the browser. The browser immediately uses the signed URL to access the file in the CloudFront edge cache without any intervention from the user.
7. CloudFront uses the public key to validate the signature and confirm that the URL hasn't been tampered with. If the signature is invalid, the request is rejected.

   If the signature is valid, CloudFront looks at the policy statement in the URL (or constructs one if you're using a canned policy) to confirm that the request is still valid. For example, if you specified a beginning and ending date and time for the URL, CloudFront confirms that the user is trying to access your content during the time period that you want to allow access.

   If the request meets the requirements in the policy statement, CloudFront does the standard operations: determines whether the file is already in the edge cache, forwards the request to the origin if necessary, and returns the file to the user.

Note

If an unsigned URL contains query string parameters, make sure you include them in the portion of the URL that you sign. If you add a query string to a signed URL after signing it, the URL returns an HTTP 403 status.

Note

CloudFront doesn't require a signed URL or signed cookies for `OPTIONS` requests. This allows CORS preflight requests, which browsers send without credentials, to succeed for cache behaviors that require signed URLs or signed cookies. If the cache behavior's allowed methods include `OPTIONS`, CloudFront forwards `OPTIONS` requests to your origin without verifying a signature. Make sure that your origin doesn't return protected content in response to `OPTIONS` requests, or restrict the allowed methods to `GET` and `HEAD` if your application doesn't use CORS.

Decide how long signed URLs are valid

You can distribute private content using a signed URL that is valid for only a short time—possibly for as little as a few minutes. Signed URLs that are valid for such a short period are good for distributing content on-the-fly to a user for a specific purpose, such as distributing movie rentals or music downloads to customers on demand. If your signed URLs will be valid for just a short period, you'll probably want to generate them automatically using an application that you develop. When the user starts to download a file or starts to play a media file, CloudFront compares the expiration time in the URL with the current time to determine whether the URL is still valid.

You can also distribute private content using a signed URL that is valid for a longer time, possibly for years. Signed URLs that are valid for a longer period are useful for distributing private content to known users, such as distributing a business plan to investors or distributing training materials to employees. You can develop an application to generate these longer-term signed URLs for you.

When CloudFront checks expiration date and time in a signed URL

CloudFront checks the expiration date and time in a signed URL at the time of the HTTP request. If a client begins to download a large file immediately before the expiration time, the download should complete even if the expiration time passes during the download. If the TCP connection

drops and the client tries to restart the download after the expiration time passes, the download will fail.

If a client uses Range GETs to get a file in smaller pieces, any GET request that occurs after the expiration time passes will fail. For more information about Range GETs, see How CloudFront processes partial requests for an object (range GETs).

Example code and third-party tools

For example code that creates the hashed and signed part of signed URLs, see the following topics:

- Create a URL signature using Perl
- Create a URL signature using PHP
- Create a URL signature using C# and the .NET Framework
- Create a URL signature using Java

Create a signed URL using a canned policy

To create a signed URL using a canned policy, complete the following steps.

To create a signed URL using a canned policy

1. If you're using .NET or Java to create signed URLs, and if you haven't reformatted the private key for your key pair from the default .pem format to a format compatible with .NET or with Java, do so now. For more information, see Reformat the private key (.NET and Java only).
2. Concatenate the following values. You can use the format in this example signed URL.

```
https://d111111abcdef8.cloudfront.net/
image.jpg?color=red&size=medium&Expires=1767290400&Signature=nitfHRCrtziwO2HwPfWw~yYDhUF5Ew
j19DzZrvDh6hQ73lDx~-ar3UocvvRQVw6EkC~GdpGQyyOSKQim-
TxAnW7d8F5Kkai9HVx0FIu-5jcQb0UEmatEXAMPLE3ReXySpLSMj0yCd3ZAB4UcBCAqEijkytL6f3fVYNGQI6&Key-
Pair-Id=K2JCJMDEHXQW5F&Hash-Algorithm=SHA256
```

   Remove all empty spaces (including tabs and newline characters). You might have to include escape characters in the string in application code. All values have a type of `String`.

1. ***Base URL for the file***

   The base URL is the CloudFront URL that you would use to access the file if you were not using signed URLs, including your own query string parameters, if any. In the preceding example, the base URL is `https://d111111abcdef8.cloudfront.net/image.jpg`. For more information about the format of URLs for distributions, see Customize the URL format for files in CloudFront.

   - The following CloudFront URL is for an image file in a distribution (using the CloudFront domain name). Note that `image.jpg` is in an `images` directory. The path to the file in the URL must match the path to the file on your HTTP server or in your Amazon S3 bucket.

     ```
     https://d111111abcdef8.cloudfront.net/images/image.jpg
     ```

   - The following CloudFront URL includes a query string:

     ```
     https://d111111abcdef8.cloudfront.net/images/image.jpg?size=large
     ```

   - The following CloudFront URLs are for image files in a distribution. Both use an alternate domain name. The second one includes a query string:

     ```
     https://www.example.com/images/image.jpg
     ```

     ```
     https://www.example.com/images/image.jpg?color=red
     ```

   - The following CloudFront URL is for an image file in a distribution that uses an alternate domain name and the HTTPS protocol:

     ```
     https://www.example.com/images/image.jpg
     ```

2. **?**

   The ? indicates that query parameters follow the base URL. Include the ? even if you don't specify any query parameters.

   > **Note**
   >
   > You can specify the following query parameters in any order.

3. ***Your query string parameters, if any*&**

   (Optional) You can enter your own query string parameters. To do so, add an ampersand (&) between each one, such as `color=red&size=medium`. You can specify query string parameters in any order within the URL.

   > **Important**
   >
   > Your query string parameters can't be named `Expires`, `Signature`, `Key-Pair-Id`, or `Hash-Algorithm`.

4. **Expires=*date and time in Unix time format (in seconds) and Coordinated Universal Time (UTC)***

   The date and time that you want the URL to stop allowing access to the file.

   Specify the expiration date and time in Unix time format (in seconds) and Coordinated Universal Time (UTC). For example, January 1, 2026 10:00 am UTC converts to `1767290400` in Unix time format, as shown in the example at the start of this topic.

   To use epoch time, specify a 64-bit integer for a date that's no later than `9223372036854775807` (Friday, April 11, 2262 at 23:47:16.854 UTC).

   For information about UTC, see RFC 3339, Date and Time on the Internet: Timestamps.

5. **&Signature=*hashed and signed version of the policy statement***

   A hashed, signed, and base64-encoded version of the JSON policy statement. For more information, see Create a signature for a signed URL that uses a canned policy.

6. **&Key-Pair-Id=*public key ID for the CloudFront public key whose corresponding private key you're using to generate the signature***

   The ID for a CloudFront public key, for example, `K2JCJMDEHXQW5F`. The public key ID tells CloudFront which public key to use to validate the signed URL. CloudFront compares the information in the signature with the information in the policy statement to verify that the URL has not been tampered with.

This public key must belong to a key group that is a trusted signer in the distribution. For more information, see Specify signers that can create signed URLs and signed cookies.

7. **`&Hash-Algorithm=`*`SHA1 or SHA256`***

   (Optional) The hash algorithm used to create the signature. Supported values are SHA1 and SHA256. If you don't specify this parameter, CloudFront defaults to SHA1.

Create a signature for a signed URL that uses a canned policy

To create the signature for a signed URL that uses a canned policy, complete the following procedures.

Topics

- Create a policy statement for a signed URL that uses a canned policy
- Create a signature for a signed URL that uses a canned policy

Create a policy statement for a signed URL that uses a canned policy

When you create a signed URL using a canned policy, the `Signature` parameter is a hashed and signed version of a policy statement. For signed URLs that use a canned policy, you don't include the policy statement in the URL, as you do for signed URLs that use a custom policy. To create the policy statement, do the following procedure.

To create the policy statement for a signed URL that uses a canned policy

1. Construct the policy statement using the following JSON format and using UTF-8 character encoding. Include all punctuation and other literal values exactly as specified. For information about the `Resource` and `DateLessThan` parameters, see Values that you specify in the policy statement for a signed URL that uses a canned policy.

```
{
    "Statement": [
        {
            "Resource": "base URL or stream name",
            "Condition": {
                "DateLessThan": {
                    "AWS:EpochTime": ending date and time in Unix time format and
 UTC
```

```
                }
            }
        }
    ]
}
```

2. Remove all empty spaces (including tabs and newline characters) from the policy statement. You might have to include escape characters in the string in application code.

Values that you specify in the policy statement for a signed URL that uses a canned policy

When you create a policy statement for a canned policy, you specify the following values.

Resource

> **Note**
>
> You can specify only one value for `Resource`.

The base URL including your query strings, if any, but excluding the CloudFront `Expires`, `Signature`, `Key-Pair-Id`, and `Hash-Algorithm` parameters, for example:

```
https://d111111abcdef8.cloudfront.net/images/horizon.jpg?
size=large&license=yes
```

Note the following:

- **Protocol** – The value must begin with `http://` or `https://`.
- **Query string parameters** – If you have no query string parameters, omit the question mark.
- **Alternate domain names** – If you specify an alternate domain name (CNAME) in the URL, you must specify the alternate domain name when referencing the file in your webpage or application. Do not specify the Amazon S3 URL for the object.
- **Special characters** – If the URL for your resource contains an asterisk (*) or question mark (?) character (for instance, in the object key name), you must URL-encode those characters (`%2A` for `*` and `%3F` for ?). This is because CloudFront interprets unencoded `*` and ? characters as wildcard characters in the resource path. If these characters are not URL-encoded, the signed URL might grant broader access than intended.

DateLessThan

The expiration date and time for the URL in Unix time format (in seconds) and Coordinated Universal Time (UTC). For example, January 1, 2026 10:00 am UTC converts to 1767290400 in Unix time format.

This value must match the value of the `Expires` query string parameter in the signed URL. Do not enclose the value in quotation marks.

For more information, see When CloudFront checks expiration date and time in a signed URL.

Example policy statement for a signed URL that uses a canned policy

When you use the following example policy statement in a signed URL, a user can access the file `https://d111111abcdef8.cloudfront.net/horizon.jpg` until January 1, 2026 10:00 am UTC:

```
{
    "Statement": [
        {
            "Resource": "https://d111111abcdef8.cloudfront.net/horizon.jpg?
size=large&license=yes",
            "Condition": {
                "DateLessThan": {
                    "AWS:EpochTime": 1767290400
                }
            }
        }
    ]
}
```

Create a signature for a signed URL that uses a canned policy

To create the value for the `Signature` parameter in a signed URL, you hash and sign the policy statement that you created in Create a policy statement for a signed URL that uses a canned policy.

For additional information and examples of how to hash, sign, and encode the policy statement, see:

- Linux commands and OpenSSL for base64 encoding and encryption
- Code examples for creating a signature for a signed URL

> **Note**
>
> The linked examples use SHA-1 by default. To use SHA-256 instead, replace `sha1` with `sha256` in the OpenSSL commands and include the `Hash-Algorithm=SHA256` query parameter in the signed URL.

Option 1: To create a signature by using a canned policy

1. Use the SHA-1 or SHA-256 hash function and the generated RSA or ECDSA private key to hash and sign the policy statement that you created in the procedure To create the policy statement for a signed URL that uses a canned policy. Use the version of the policy statement that no longer includes empty spaces.

   If you use SHA-256, you must include `&Hash-Algorithm=SHA256` in the signed URL.

   For the private key that is required by the hash function, use a private key whose public key is in an active trusted key group for the distribution.

   > **Note**
   >
   > The method that you use to hash and sign the policy statement depends on your programming language and platform. For sample code, see Code examples for creating a signature for a signed URL.

2. Remove empty spaces (including tabs and newline characters) from the hashed and signed string.
3. Base64-encode the string using MIME base64 encoding. For more information, see Section 6.8, Base64 Content-Transfer-Encoding in *RFC 2045, MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*.
4. Replace characters that are invalid in a URL query string with characters that are valid. The following table lists invalid and valid characters.

Replace these invalid characters	With these valid characters
+	- (hyphen)

Replace these invalid characters	With these valid characters
=	_ (underscore)
/	~ (tilde)

5. Append the resulting value to your signed URL after `&Signature=`, and return to To create a signed URL using a canned policy to finish concatenating the parts of your signed URL.

Create a signed URL using a custom policy

To create a signed URL using a custom policy, complete the following procedure.

To create a signed URL using a custom policy

1. If you're using .NET or Java to create signed URLs, and if you haven't reformatted the private key for your key pair from the default .pem format to a format compatible with .NET or with Java, do so now. For more information, see Reformat the private key (.NET and Java only).
2. Concatenate the following values. You can use the format in this example signed URL.

```
https://d111111abcdef8.cloudfront.net/
image.jpg?color=red&size=medium&Policy=eyANCiAgICEXAMPLEW1lbnQiOiBbeyANCiAgICAgICJSZXNvdXJj
j19DzZrvDh6hQ73lDx~-ar3UocvvRQVw6EkC~GdpGQyyOSKQim-
TxAnW7d8F5Kkai9HVx0FIu-5jcQb0UEmatEXAMPLE3ReXySpLSMj0yCd3ZAB4UcBCAqEijkytL6f3fVYNGQI6&Key-
Pair-Id=K2JCJMDEHXQW5F&Hash-Algorithm=SHA256
```

Remove all empty spaces (including tabs and newline characters). You might have to include escape characters in the string in application code. All values have a type of `String`.

1. ***Base URL for the file***

   The base URL is the CloudFront URL that you would use to access the file if you were not using signed URLs, including your own query string parameters, if any. In the preceding example, the base URL is `https://d111111abcdef8.cloudfront.net/image.jpg`. For more information about the format of URLs for distributions, see Customize the URL format for files in CloudFront.

   The following examples show values that you specify for distributions.

- The following CloudFront URL is for an image file in a distribution (using the CloudFront domain name). Note that `image.jpg` is in an `images` directory. The path to the file in the URL must match the path to the file on your HTTP server or in your Amazon S3 bucket.

  ```
  https://d111111abcdef8.cloudfront.net/images/image.jpg
  ```

- The following CloudFront URL includes a query string:

  ```
  https://d111111abcdef8.cloudfront.net/images/image.jpg?size=large
  ```

- The following CloudFront URLs are for image files in a distribution. Both use an alternate domain name; the second one includes a query string:

  ```
  https://www.example.com/images/image.jpg
  ```

  ```
  https://www.example.com/images/image.jpg?color=red
  ```

- The following CloudFront URL is for an image file in a distribution that uses an alternate domain name and the HTTPS protocol:

  ```
  https://www.example.com/images/image.jpg
  ```

2. ?

The ? indicates that query string parameters follow the base URL. Include the ? even if you don't specify any query parameters.

> **Note**
>
> You can specify the following query parameters in any order.

3. *Your query string parameters, if any*&

(Optional) You can enter your own query string parameters. To do so, add an ampersand (&) between each one, such as `color=red&size=medium`. You can specify query string parameters in any order within the URL.

> **Important**
>
> Your query string parameters can't be named `Policy`, `Signature`, `Key-Pair-Id`, or `Hash-Algorithm`.

If you add your own parameters, append an & after each one, including the last one.

4. **`Policy=`*`base64 encoded version of policy statement`***

   Your policy statement in JSON format, with empty spaces removed, then base64 encoded. For more information, see Create a policy statement for a signed URL that uses a custom policy.

   The policy statement controls the access that a signed URL grants to a user. It includes the URL of the file, an expiration date and time, an optional date and time that the URL becomes valid, and an optional IP address or range of IP addresses that are allowed to access the file.

5. **`&Signature=`*`hashed and signed version of the policy statement`***

   A hashed, signed, and base64-encoded version of the JSON policy statement. For more information, see Create a signature for a signed URL that uses a custom policy.

6. **`&Key-Pair-Id=`*`public key ID for the CloudFront public key whose corresponding private key you're using to generate the signature`***

   The ID for a CloudFront public key, for example, `K2JCJMDEHXQW5F`. The public key ID tells CloudFront which public key to use to validate the signed URL. CloudFront compares the information in the signature with the information in the policy statement to verify that the URL has not been tampered with.

   This public key must belong to a key group that is a trusted signer in the distribution. For more information, see Specify signers that can create signed URLs and signed cookies.

7. **`&Hash-Algorithm=`*`SHA1 or SHA256`***

   (Optional) The hash algorithm used to create the signature. Supported values are `SHA1` and `SHA256`. If you don't specify this parameter, CloudFront defaults to `SHA1`.

Create a policy statement for a signed URL that uses a custom policy

Complete the following steps to create a policy statement for a signed URL that uses a custom policy.

For example policy statements that control access to files in a variety of ways, see the section called “Example policy statements for a signed URL that uses a custom policy”.

To create the policy statement for a signed URL that uses a custom policy

1. Construct the policy statement using the following JSON format. Replace the less than (<) and greater than (>) symbols, and the descriptions within them, with your own values. For more information, see the section called “Values that you specify in the policy statement for a signed URL that uses a custom policy”.

```
{
    "Statement": [
        {
            "Resource": "<Optional but recommended: URL of the file>",
            "Condition": {
                "DateLessThan": {
                 "AWS:EpochTime": <Required: ending date and time in Unix time
 format and UTC>
                },
                "DateGreaterThan": {
                 "AWS:EpochTime": <Optional: beginning date and time in Unix time
 format and UTC>
                },
                "IpAddress": {
                 "AWS:SourceIp": "<Optional: IP address>"
                }
            }
        }
    ]
}
```

   Note the following:

   - You can include only one statement in the policy.
   - Use UTF-8 character encoding.

- Include all punctuation and parameter names exactly as specified. Abbreviations for parameter names are not accepted.
- The order of the parameters in the `Condition` section doesn't matter.
- For information about the values for `Resource`, `DateLessThan`, `DateGreaterThan`, and `IpAddress`, see the section called "Values that you specify in the policy statement for a signed URL that uses a custom policy".

2. Remove all empty spaces (including tabs and newline characters) from the policy statement. You might have to include escape characters in the string in application code.
3. Base64-encode the policy statement using MIME base64 encoding. For more information, see Section 6.8, Base64 Content-Transfer-Encoding in *RFC 2045, MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*.
4. Replace characters that are invalid in a URL query string with characters that are valid. The following table lists invalid and valid characters.

Replace these invalid characters	With these valid characters
+	- (hyphen)
=	_ (underscore)
/	~ (tilde)

5. Append the resulting value to your signed URL after `Policy=`.
6. Create a signature for the signed URL by hashing, signing, and base64-encoding the policy statement. For more information, see the section called "Create a signature for a signed URL that uses a custom policy".

Values that you specify in the policy statement for a signed URL that uses a custom policy

When you create a policy statement for a custom policy, you specify the following values.

Resource

The URL, including any query strings, but excluding the CloudFront `Policy`, `Signature`, `Key-Pair-Id`, and `Hash-Algorithm` parameters. For example:

```
https://d111111abcdef8.cloudfront.net/images/horizon.jpg\?
size=large&license=yes
```

You can specify only one URL value for `Resource`.

> **Important**
>
> You can omit the `Resource` parameter in a policy, but doing so means that anyone with the signed URL can access *all* of the files in *any* distribution that is associated with the key pair that you use to create the signed URL.

Note the following:

- **Protocol** – The value must begin with `http://`, `https://`, or `*://`.
- **Query string parameters** – If the URL has query string parameters, don't use a backslash character (\) to escape the question mark character (?) that begins the query string. For example:

  ```
  https://d111111abcdef8.cloudfront.net/images/horizon.jpg?
  size=large&license=yes
  ```
- **Wildcard characters** – You can use wildcard characters in the URL in the policy. The following wildcard characters are supported:
  - asterisk (*), which matches zero or more characters
  - question mark (?), which matches exactly one character

  When CloudFront matches the URL in the policy to the URL in the HTTP request, the URL in the policy is divided into four sections—protocol, domain, path, and query string—as follows:

  ```
  [protocol]://[domain]/[path]\?[query string]
  ```

  When you use a wildcard character in the URL in the policy, the wildcard matching applies only within the boundaries of the section that contains the wildcard. For example, consider this URL in a policy:

  ```
  https://www.example.com/hello*world
  ```

In this example, the asterisk wildcard (*) only applies within the path section, so it matches the URLs `https://www.example.com/helloworld` and `https://www.example.com/hello-world`, but it does not match the URL `https://www.example.net/hello?world`.

The following exceptions apply to the section boundaries for wildcard matching:

- A trailing asterisk in the path section implies an asterisk in the query string section. For example, `http://example.com/hello*` is equivalent to `http://example.com/hello*\?*`.
- A trailing asterisk in the domain section implies an asterisk in both the path and query string sections. For example, `http://example.com*` is equivalent to `http://example.com*/*\?*`.
- A URL in the policy can omit the protocol section and start with an asterisk in the domain section. In that case, the protocol section is implicitly set to an asterisk. For example, the URL `*example.com` in a policy is equivalent to `*://*example.com/`.
- An asterisk by itself (`"Resource": "*"`) matches any URL.

For example, the value: `https://d111111abcdef8.cloudfront.net/*game_download.zip*` in a policy matches all of the following URLs:

- `https://d111111abcdef8.cloudfront.net/game_download.zip`
- `https://d111111abcdef8.cloudfront.net/example_game_download.zip?license=yes`
- `https://d111111abcdef8.cloudfront.net/test_game_download.zip?license=temp`

Percent-encoded characters – CloudFront matches the policy `Resource` against the raw (undecoded) request URL and forwards that URL to the origin unchanged. A percent-encoded character in the request URL, such as `%3B` for `;`, `%3F` for `?`, or `%23` for `#`, is treated as data rather than as a URL delimiter. For more information, see RFC 3986 on the IETF website. For example, a wildcard can't match across a literal semicolon (`;`), but it can match across the encoded form (`%3B`).

- **Alternate domain names** – If you specify an alternate domain name (CNAME) in the URL in the policy, the HTTP request must use the alternate domain name in your webpage or application. Do not specify the Amazon S3 URL for the file in a policy.

DateLessThan

The expiration date and time for the URL in Unix time format (in seconds) and Coordinated Universal Time (UTC). In the policy, do not enclose the value in quotation marks. For information about UTC, see Date and Time on the Internet: Timestamps.

For example, January 31, 2023 10:00 AM UTC converts to 1675159200 in Unix time format.

This is the only required parameter in the `Condition` section. CloudFront requires this value to prevent users from having permanent access to your private content.

For more information, see the section called "When CloudFront checks expiration date and time in a signed URL"

DateGreaterThan (Optional)

An optional start date and time for the URL in Unix time format (in seconds) and Coordinated Universal Time (UTC). Users are not allowed to access the file on or before the specified date and time. Do not enclose the value in quotation marks.

IpAddress (Optional)

The IP address of the client making the HTTP request. Note the following:

- To allow any IP address to access the file, omit the `IpAddress` parameter.
- You can specify either one IP address or one IP address range. You can't use the policy to allow access if the client's IP address is in one of two separate ranges.
- To allow access from a single IP address, you specify:

  `"IPv4 IP address/32"`
- You must specify IP address ranges in standard IPv4 CIDR format (for example, `192.0.2.0/24`). For more information, see Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan.

> **Important**
>
> IP addresses in IPv6 format, such as 2001:0db8:85a3::8a2e:0370:7334, are not supported.

If you're using a custom policy that includes `IpAddress`, do not enable IPv6 for the distribution. If you want to restrict access to some content by IP address and support IPv6

requests for other content, you can create two distributions. For more information, see the section called "Enable IPv6 (viewer requests)" in the topic the section called "All distribution settings".

Example policy statements for a signed URL that uses a custom policy

The following example policy statements show how to control access to a specific file, all of the files in a directory, or all of the files associated with a key pair ID. The examples also show how to control access from an individual IP address or a range of IP addresses, and how to prevent users from using the signed URL after a specified date and time.

If you copy and paste any of these examples, remove any empty spaces (including tabs and newline characters), replace the values with your own values, and include a newline character after the closing brace (}).

For more information, see the section called "Values that you specify in the policy statement for a signed URL that uses a custom policy".

Topics

- Example policy statement: Access one file from a range of IP addresses
- Example policy statement: Access all files in a directory from a range of IP addresses
- Example policy statement: Access all files associated with a key pair ID from one IP address

Example policy statement: Access one file from a range of IP addresses

The following example custom policy in a signed URL specifies that a user can access the file `https://d111111abcdef8.cloudfront.net/game_download.zip` from IP addresses in the range `192.0.2.0/24` until January 31, 2023 10:00 AM UTC:

```
{
    "Statement": [
        {
            "Resource": "https://d111111abcdef8.cloudfront.net/game_download.zip",
            "Condition": {
                "IpAddress": {
                    "AWS:SourceIp": "192.0.2.0/24"
                },
                "DateLessThan": {
                    "AWS:EpochTime": 1675159200
```

```
                }
            }
        }
    ]
}
```

Example policy statement: Access all files in a directory from a range of IP addresses

With the following example custom policy, you can create signed URLs for any file in the `training` directory, as indicated by the asterisk wildcard character (*) in the `Resource` parameter. Users can access the file from an IP address in the range `192.0.2.0/24` until January 31, 2023 10:00 AM UTC:

```
{
    "Statement": [
        {
            "Resource": "https://d111111abcdef8.cloudfront.net/training/*",
            "Condition": {
                "IpAddress": {
                    "AWS:SourceIp": "192.0.2.0/24"
                },
                "DateLessThan": {
                    "AWS:EpochTime": 1675159200
                }
            }
        }
    ]
}
```

Each signed URL with which you use this policy has a URL that identifies a specific file, for example:

`https://d111111abcdef8.cloudfront.net/training/orientation.pdf`

Example policy statement: Access all files associated with a key pair ID from one IP address

With the following example custom policy, you can create signed URLs for any file associated with any distribution, as indicated by the asterisk wildcard character (*) in the `Resource` parameter. The signed URL must use the `https://` protocol, not `http://`. The user must use the IP address `192.0.2.10/32`. (The value `192.0.2.10/32` in CIDR notation refers to a single IP address, `192.0.2.10`.) The files are available only from January 31, 2023 10:00 AM UTC until February 2, 2023 10:00 AM UTC:

```
{
    "Statement": [
       {
            "Resource": "https://*",
            "Condition": {
                "IpAddress": {
                    "AWS:SourceIp": "192.0.2.10/32"
                },
                "DateGreaterThan": {
                    "AWS:EpochTime": 1675159200
                },
                "DateLessThan": {
                    "AWS:EpochTime": 1675332000
                }
            }
         }
    ]
}
```

Each signed URL with which you use this policy has a URL that identifies a specific file in a specific CloudFront distribution, for example:

`https://d111111abcdef8.cloudfront.net/training/orientation.pdf`

The signed URL also includes a key pair ID, which must be associated with a trusted key group in the distribution (d111111abcdef8.cloudfront.net) that you specify in the URL.

Create a signature for a signed URL that uses a custom policy

The signature for a signed URL that uses a custom policy is a hashed, signed, and base64-encoded version of the policy statement. To create a signature for a custom policy, complete the following steps.

For additional information and examples of how to hash, sign, and encode the policy statement, see:

- Linux commands and OpenSSL for base64 encoding and encryption
- Code examples for creating a signature for a signed URL

> **Note**
>
> The linked examples use SHA-1 by default. To use SHA-256 instead, replace `sha1` with `sha256` in the OpenSSL commands and include the `Hash-Algorithm=SHA256` query parameter in the signed URL.

Option 1: To create a signature by using a custom policy

1. Use the SHA-1 or SHA-256 hash function and the generated RSA or ECDSA private key to hash and sign the JSON policy statement that you created in the procedure To create the policy statement for a signed URL that uses a custom policy. Use the version of the policy statement that no longer includes empty spaces but that has not yet been base64-encoded.

   If you use SHA-256, you must include `&Hash-Algorithm=SHA256` in the signed URL.

   For the private key that is required by the hash function, use a private key whose public key is in an active trusted key group for the distribution.

   > **Note**
   >
   > The method that you use to hash and sign the policy statement depends on your programming language and platform. For sample code, see Code examples for creating a signature for a signed URL.

2. Remove empty spaces (including tabs and newline characters) from the hashed and signed string.
3. Base64-encode the string using MIME base64 encoding. For more information, see Section 6.8, Base64 Content-Transfer-Encoding in *RFC 2045, MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*.
4. Replace characters that are invalid in a URL query string with characters that are valid. The following table lists invalid and valid characters.

Replace these invalid characters	With these valid characters
+	- (hyphen)

Replace these invalid characters	With these valid characters
=	_ (underscore)
/	~ (tilde)

5. Append the resulting value to your signed URL after `&Signature=`, and return to To create a signed URL using a custom policy to finish concatenating the parts of your signed URL.

Use signed cookies

CloudFront signed cookies allow you to control who can access your content when you don't want to change your current URLs or when you want to provide access to multiple restricted files, for example, all of the files in the subscribers' area of a website. This topic explains the considerations when using signed cookies and describes how to set signed cookies using canned and custom policies.

Topics

- Decide to use canned or custom policies for signed cookies
- How signed cookies work
- Prevent misuse of signed cookies
- When CloudFront checks expiration date and time in a signed cookie
- Sample code and third-party tools
- Set signed cookies using a canned policy
- Set signed cookies using a custom policy
- Create signed cookies using PHP

Decide to use canned or custom policies for signed cookies

When you create a signed cookie, you write a policy statement in JSON format that specifies the restrictions on the signed cookie, for example, how long the cookie is valid. You can use canned policies or custom policies. The following table compares canned and custom policies:

Description	Canned policy	Custom policy
You can reuse the policy statement for multiple files. To reuse the policy statement, you must use wildcard characters in the `Resource` object. For more information, see Values that you specify in the policy statement for a custom policy for signed cookies.)	No	Yes
You can specify the date and time that users can begin to access your content	No	Yes (optional)
You can specify the date and time that users can no longer access your content	Yes	Yes
You can specify the IP address or range of IP addresses of the users who can access your content	No	Yes (optional)

For information about creating signed cookies using a canned policy, see Set signed cookies using a canned policy.

For information about creating signed cookies using a custom policy, see Set signed cookies using a custom policy.

How signed cookies work

Here's an overview of how you configure CloudFront for signed cookies and how CloudFront responds when a user submits a request that contains a signed cookie.

1. In your CloudFront distribution, specify one or more trusted key groups, which contain the public keys that CloudFront can use to verify the URL signature. You use the corresponding private keys to sign the URLs.

   For more information, see Specify signers that can create signed URLs and signed cookies.
2. You develop your application to determine whether a user should have access to your content and, if so, to send three `Set-Cookie` headers to the viewer. (Each `Set-Cookie` header can contain only one name-value pair, and a CloudFront signed cookie requires three name-value pairs.) You must send the `Set-Cookie` headers to the viewer before the viewer requests

your private content. If you set a short expiration time on the cookie, you might also want to send three more `Set-Cookie` headers in response to subsequent requests, so that the user continues to have access.

Typically, your CloudFront distribution will have at least two cache behaviors, one that doesn't require authentication and one that does. The error page for the secure portion of the site includes a redirector or a link to a login page.

If you configure your distribution to cache files based on cookies, CloudFront doesn't cache separate files based on the attributes in signed cookies.

3. A user signs in to your website and either pays for content or meets some other requirement for access.
4. Your application returns the `Set-Cookie` headers in the response, and the viewer stores the name-value pairs.
5. The user requests a file.

   The user's browser or other viewer gets the name-value pairs from step 4 and adds them to the request in a `Cookie` header. This is the signed cookie.
6. CloudFront uses the public key to validate the signature in the signed cookie and to confirm that the cookie hasn't been tampered with. If the signature is invalid, the request is rejected.

   If the signature in the cookie is valid, CloudFront looks at the policy statement in the cookie (or constructs one if you're using a canned policy) to confirm that the request is still valid. For example, if you specified a beginning and ending date and time for the cookie, CloudFront confirms that the user is trying to access your content during the time period that you want to allow access.

   If the request meets the requirements in the policy statement, CloudFront serves your content as it does for content that isn't restricted: it determines whether the file is already in the edge cache, forwards the request to the origin if necessary, and returns the file to the user.

Note

CloudFront doesn't require a signed URL or signed cookies for `OPTIONS` requests. This allows CORS preflight requests, which browsers send without credentials, to succeed for cache behaviors that require signed URLs or signed cookies. If the cache behavior's allowed methods include `OPTIONS`, CloudFront forwards `OPTIONS` requests to your origin

without verifying a signature. Make sure that your origin doesn't return protected content in response to `OPTIONS` requests, or restrict the allowed methods to `GET` and `HEAD` if your application doesn't use CORS.

Prevent misuse of signed cookies

If you specify the `Domain` parameter in a `Set-Cookie` header, specify the most precise value possible to reduce the potential for access by someone with the same root domain name. For example, app.example.com is preferable to example.com, especially when you don't control example.com. This helps prevent someone from accessing your content from www.example.com.

To help prevent this type of attack, do the following:

- Exclude the `Expires` and `Max-Age` cookie attributes, so that the `Set-Cookie` header creates a session cookie. Session cookies are automatically deleted when the user closes the browser, which reduces the possibility of someone getting unauthorized access to your content.
- Include the `Secure` attribute, so that the cookie is encrypted when a viewer includes it in a request.
- When possible, use a custom policy and include the IP address of the viewer.
- In the `CloudFront-Expires` attribute, specify the shortest reasonable expiration time based on how long you want users to have access to your content.

When CloudFront checks expiration date and time in a signed cookie

To determine whether a signed cookie is still valid, CloudFront checks the expiration date and time in the cookie at the time of the HTTP request. If a client begins to download a large file immediately before the expiration time, the download should complete even if the expiration time passes during the download. If the TCP connection drops and the client tries to restart the download after the expiration time passes, the download will fail.

If a client uses Range GETs to get a file in smaller pieces, any GET request that occurs after the expiration time passes will fail. For more information about Range GETs, see How CloudFront processes partial requests for an object (range GETs).

Sample code and third-party tools

The sample code for private content shows only how to create the signature for signed URLs. However, the process for creating a signature for a signed cookie is very similar, so much of the sample code is still relevant. For more information, see the following topics:

- Create a URL signature using Perl
- Create a URL signature using PHP
- Create a URL signature using C# and the .NET Framework
- Create a URL signature using Java

Set signed cookies using a canned policy

To set a signed cookie by using a canned policy, complete the following steps. To create the signature, see Create a signature for a signed cookie that uses a canned policy.

To set a signed cookie using a canned policy

1. If you're using .NET or Java to create signed cookies, and if you haven't reformatted the private key for your key pair from the default .pem format to a format compatible with .NET or with Java, do so now. For more information, see Reformat the private key (.NET and Java only).
2. Program your application to send three `Set-Cookie` headers to approved viewers (or four, if you want to specify a hash algorithm). You need three `Set-Cookie` headers because each `Set-Cookie` header can contain only one name-value pair, and a CloudFront signed cookie requires three name-value pairs. The name-value pairs are: `CloudFront-Expires`, `CloudFront-Signature`, and `CloudFront-Key-Pair-Id`. You can optionally include a fourth name-value pair, `CloudFront-Hash-Algorithm`, to specify the hash algorithm used for the signature. The values must be present on the viewer before a user makes the first request for a file that you want to control access to.

   > **Note**
   >
   > In general, we recommend that you exclude `Expires` and `Max-Age` attributes. Excluding the attributes causes the browser to delete the cookie when the user closes the browser, which reduces the possibility of someone getting unauthorized access to your content. For more information, see Prevent misuse of signed cookies.

The names of cookie attributes are case-sensitive.

Line breaks are included only to make the attributes more readable.

```
Set-Cookie:
CloudFront-Expires=date and time in Unix time format (in seconds) and Coordinated
 Universal Time (UTC);
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly

Set-Cookie:
CloudFront-Signature=hashed and signed version of the policy statement;
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly

Set-Cookie:
CloudFront-Key-Pair-Id=public key ID for the CloudFront public key whose
 corresponding private key you're using to generate the signature;
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly

Set-Cookie:
CloudFront-Hash-Algorithm=SHA1 or SHA256;
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly
```

(Optional) `Domain`

The domain name for the requested file. If you don't specify a `Domain` attribute, the default value is the domain name in the URL, and it applies only to the specified domain name, not to subdomains. If you specify a `Domain` attribute, it also applies to subdomains. A leading dot in the domain name (for example, `Domain=.example.com`) is optional. In

addition, if you specify a `Domain` attribute, the domain name in the URL and the value of the `Domain` attribute must match.

You can specify the domain name that CloudFront assigned to your distribution, for example, d111111abcdef8.cloudfront.net, but you can't specify *.cloudfront.net for the domain name.

If you want to use an alternate domain name such as example.com in URLs, you must add the alternate domain name to your distribution regardless of whether you specify the `Domain` attribute. For more information, see Alternate domain names (CNAMEs) in the topic All distribution settings reference.

(Optional) `Path`

The path for the requested file. If you don't specify a `Path` attribute, the default value is the path in the URL.

`Secure`

Requires that the viewer encrypt cookies before sending a request. We recommend that you send the `Set-Cookie` header over an HTTPS connection to make sure that the cookie attributes are protected from man-in-the-middle attacks.

`HttpOnly`

Defines how the browser (where supported) interacts with the cookie value. With `HttpOnly`, the cookie values are inaccessible to JavaScript. This precaution can help mitigate cross-site scripting (XSS) attacks. For more information, see Using HTTP cookies.

`CloudFront-Expires`

Specify the expiration date and time in Unix time format (in seconds) and Coordinated Universal Time (UTC). For example, January 1, 2026 10:00 am UTC converts to 1767290400 in Unix time format.

To use epoch time, specify a 64-bit integer for a date that's no later than `9223372036854775807` (Friday, April 11, 2262 at 23:47:16.854 UTC).

For information about UTC, see *RFC 3339, Date and Time on the Internet: Timestamps*, https://tools.ietf.org/html/rfc3339.

`CloudFront-Signature`

A hashed, signed, and base64-encoded version of a JSON policy statement. For more information, see Create a signature for a signed cookie that uses a canned policy.

`CloudFront-Key-Pair-Id`

The ID for a CloudFront public key, for example, `K2JCJMDEHXQW5F`. The public key ID tells CloudFront which public key to use to validate the signed URL. CloudFront compares the information in the signature with the information in the policy statement to verify that the URL has not been tampered with.

This public key must belong to a key group that is a trusted signer in the distribution. For more information, see Specify signers that can create signed URLs and signed cookies.

`CloudFront-Hash-Algorithm`

(Optional) The hash algorithm used to create the signature. Supported values are SHA1 and SHA256. If you don't include this cookie, CloudFront defaults to SHA1.

The following example shows `Set-Cookie` headers for one signed cookie when you're using the domain name that is associated with your distribution in the URLs for your files:

```
Set-Cookie: CloudFront-Expires=1426500000; Domain=d111111abcdef8.cloudfront.net; Path=/
images/*; Secure; HttpOnly
Set-Cookie: CloudFront-Signature=yXrSIgyQoeE4FBI4eMKF6ho~CA8_;
 Domain=d111111abcdef8.cloudfront.net; Path=/images/*; Secure; HttpOnly
Set-Cookie: CloudFront-Key-Pair-Id=K2JCJMDEHXQW5F;
 Domain=d111111abcdef8.cloudfront.net; Path=/images/*; Secure; HttpOnly
Set-Cookie: CloudFront-Hash-Algorithm=SHA256; Domain=d111111abcdef8.cloudfront.net;
 Path=/images/*; Secure; HttpOnly
```

The following example shows `Set-Cookie` headers for one signed cookie when you're using the alternate domain name example.org in the URLs for your files:

```
Set-Cookie: CloudFront-Expires=1426500000; Domain=example.org; Path=/images/*; Secure;
 HttpOnly
Set-Cookie: CloudFront-Signature=yXrSIgyQoeE4FBI4eMKF6ho~CA8_; Domain=example.org;
 Path=/images/*; Secure; HttpOnly
Set-Cookie: CloudFront-Key-Pair-Id=K2JCJMDEHXQW5F; Domain=example.org; Path=/images/*;
 Secure; HttpOnly
```

```
Set-Cookie: CloudFront-Hash-Algorithm=SHA256; Domain=example.org; Path=/images/*;
 Secure; HttpOnly
```

If you want to use an alternate domain name such as example.com in URLs, you must add the alternate domain name to your distribution regardless of whether you specify the `Domain` attribute. For more information, see Alternate domain names (CNAMEs) in the topic All distribution settings reference.

Create a signature for a signed cookie that uses a canned policy

To create the signature for a signed cookie that uses a canned policy, complete the following procedures.

Topics

- Create a policy statement for a signed cookie that uses a canned policy
- Sign the policy statement to create a signature for a signed cookie that uses a canned policy

Create a policy statement for a signed cookie that uses a canned policy

When you set a signed cookie that uses a canned policy, the `CloudFront-Signature` attribute is a hashed and signed version of a policy statement. For signed cookies that use a canned policy, you don't include the policy statement in the `Set-Cookie` header, as you do for signed cookies that use a custom policy. To create the policy statement, complete the following steps.

To create a policy statement for a signed cookie that uses a canned policy

1. Construct the policy statement using the following JSON format and using UTF-8 character encoding. Include all punctuation and other literal values exactly as specified. For information about the `Resource` and `DateLessThan` parameters, see Values that you specify in the policy statement for a canned policy for signed cookies.

   ```
   {
       "Statement": [
           {
               "Resource": "base URL or stream name",
               "Condition": {
                   "DateLessThan": {
                       "AWS:EpochTime": ending date and time in Unix time format and
    UTC
   ```

```
                }
            }
        }
    ]
}
```

2. Remove all empty spaces (including tabs and newline characters) from the policy statement. You might have to include escape characters in the string in application code.

Values that you specify in the policy statement for a canned policy for signed cookies

When you create a policy statement for a canned policy, you specify the following values:

Resource

The base URL including your query strings, if any, for example:

`https://d111111abcdef8.cloudfront.net/images/horizon.jpg?size=large&license=yes`

You can specify only one value for `Resource`.

Note the following:

- **Protocol** – The value must begin with `http://` or `https://`.
- **Query string parameters** – If you have no query string parameters, omit the question mark.
- **Alternate domain names** – If you specify an alternate domain name (CNAME) in the URL, you must specify the alternate domain name when referencing the file in your webpage or application. Do not specify the Amazon S3 URL for the file.

DateLessThan

The expiration date and time for the URL in Unix time format (in seconds) and Coordinated Universal Time (UTC). Do not enclose the value in quotation marks.

For example, March 16, 2015 10:00 am UTC converts to 1426500000 in Unix time format.

This value must match the value of the `CloudFront-Expires` attribute in the `Set-Cookie` header. Do not enclose the value in quotation marks.

For more information, see When CloudFront checks expiration date and time in a signed cookie.

Example policy statement for a canned policy

When you use the following example policy statement in a signed cookie, a user can access the file `https://d111111abcdef8.cloudfront.net/horizon.jpg` until March 16, 2015 10:00 am UTC:

```
{
    "Statement": [
        {
            "Resource": "https://d111111abcdef8.cloudfront.net/horizon.jpg?
size=large&license=yes",
            "Condition": {
                "DateLessThan": {
                    "AWS:EpochTime": 1426500000
                }
            }
        }
    ]
}
```

Sign the policy statement to create a signature for a signed cookie that uses a canned policy

To create the value for the `CloudFront-Signature` attribute in a `Set-Cookie` header, you hash and sign the policy statement that you created in To create a policy statement for a signed cookie that uses a canned policy.

For additional information and examples of how to hash, sign, and encode the policy statement, see the following topics:

- Linux commands and OpenSSL for base64 encoding and encryption
- Code examples for creating a signature for a signed URL

> **Note**
>
> The linked examples use SHA-1 by default. To use SHA-256 instead, replace `sha1` with `sha256` in the OpenSSL commands and include the `CloudFront-Hash-Algorithm` cookie with a value of SHA256.

To create a signature for a signed cookie using a canned policy

1. Use the SHA-1 or SHA-256 hash function and RSA to hash and sign the policy statement that you created in the procedure To create a policy statement for a signed cookie that uses a canned policy. Use the version of the policy statement that no longer includes empty spaces.

   If you use SHA-256, you must include the `CloudFront-Hash-Algorithm` cookie with a value of `SHA256`.

   For the private key that is required by the hash function, use a private key whose public key is in an active trusted key group for the distribution.

   > **Note**
   >
   > The method that you use to hash and sign the policy statement depends on your programming language and platform. For sample code, see Code examples for creating a signature for a signed URL.

2. Remove empty spaces (including tabs and newline characters) from the hashed and signed string.
3. Base64-encode the string using MIME base64 encoding. For more information, see Section 6.8, Base64 Content-Transfer-Encoding in *RFC 2045, MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*.
4. Replace characters that are invalid in a URL query string with characters that are valid. The following table lists invalid and valid characters.

Replace these invalid characters	With these valid characters
+	- (hyphen)
=	_ (underscore)
/	~ (tilde)

5. Include the resulting value in the `Set-Cookie` header for the `CloudFront-Signature` name-value pair. Then return to To set a signed cookie using a canned policy add the `Set-Cookie` header for `CloudFront-Key-Pair-Id`.

Set signed cookies using a custom policy

To set a signed cookie that uses a custom policy, complete the following steps.

To set a signed cookie using a custom policy

1. If you're using .NET or Java to create signed URLs, and if you haven't reformatted the private key for your key pair from the default .pem format to a format compatible with .NET or with Java, do so now. For more information, see Reformat the private key (.NET and Java only).
2. Program your application to send three `Set-Cookie` headers to approved viewers (or four, if you want to specify a hash algorithm). You need three `Set-Cookie` headers because each `Set-Cookie` header can contain only one name-value pair, and a CloudFront signed cookie requires three name-value pairs. The name-value pairs are: `CloudFront-Policy`, `CloudFront-Signature`, and `CloudFront-Key-Pair-Id`. You can optionally include a fourth name-value pair, `CloudFront-Hash-Algorithm`, to specify the hash algorithm used for the signature. The values must be present on the viewer before a user makes the first request for a file that you want to control access to.

   > **Note**
   >
   > In general, we recommend that you exclude `Expires` and `Max-Age` attributes. This causes the browser to delete the cookie when the user closes the browser, which reduces the possibility of someone getting unauthorized access to your content. For more information, see Prevent misuse of signed cookies.

   The names of cookie attributes are case-sensitive.

   Line breaks are included only to make the attributes more readable.

```
Set-Cookie:
CloudFront-Policy=base64 encoded version of the policy statement;
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly


Set-Cookie:
```

```
CloudFront-Signature=hashed and signed version of the policy statement;
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly

Set-Cookie:
CloudFront-Key-Pair-Id=public key ID for the CloudFront public key whose
 corresponding private key you're using to generate the signature;
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly

Set-Cookie:
CloudFront-Hash-Algorithm=SHA1 or SHA256;
Domain=optional domain name;
Path=/optional directory path;
Secure;
HttpOnly
```

(Optional) `Domain`

The domain name for the requested file. If you don't specify a `Domain` attribute, the default value is the domain name in the URL, and it applies only to the specified domain name, not to subdomains. If you specify a `Domain` attribute, it also applies to subdomains. A leading dot in the domain name (for example, `Domain=.example.com`) is optional. In addition, if you specify a `Domain` attribute, the domain name in the URL and the value of the `Domain` attribute must match.

You can specify the domain name that CloudFront assigned to your distribution, for example, d111111abcdef8.cloudfront.net, but you can't specify *.cloudfront.net for the domain name.

If you want to use an alternate domain name such as example.com in URLs, you must add the alternate domain name to your distribution regardless of whether you specify the `Domain` attribute. For more information, see Alternate domain names (CNAMEs) in the topic All distribution settings reference.

(Optional) `Path`

The path for the requested file. If you don't specify a `Path` attribute, the default value is the path in the URL.

`Secure`

Requires that the viewer encrypt cookies before sending a request. We recommend that you send the `Set-Cookie` header over an HTTPS connection to make sure that the cookie attributes are protected from man-in-the-middle attacks.

`HttpOnly`

Requires that the viewer send the cookie only in HTTP or HTTPS requests.

`CloudFront-Policy`

Your policy statement in JSON format, with empty spaces removed, then base64 encoded. For more information, see Create a signature for a signed cookie that uses a custom policy.

The policy statement controls the access that a signed cookie grants to a user. It includes the files that the user can access, an expiration date and time, an optional date and time that the URL becomes valid, and an optional IP address or range of IP addresses that are allowed to access the file.

`CloudFront-Signature`

A hashed, signed, and base64-encoded version of the JSON policy statement. For more information, see Create a signature for a signed cookie that uses a custom policy.

`CloudFront-Key-Pair-Id`

The ID for a CloudFront public key, for example, `K2JCJMDEHXQW5F`. The public key ID tells CloudFront which public key to use to validate the signed URL. CloudFront compares the information in the signature with the information in the policy statement to verify that the URL has not been tampered with.

This public key must belong to a key group that is a trusted signer in the distribution. For more information, see Specify signers that can create signed URLs and signed cookies.

`CloudFront-Hash-Algorithm`

(Optional) The hash algorithm used to create the signature. Supported values are SHA1 and SHA256. If you don't include this cookie, CloudFront defaults to SHA1.

Example `Set-Cookie` headers for custom policies

See the following examples of `Set-Cookie` header pairs.

If you want to use an alternate domain name such as example.org in URLs, you must add the alternate domain name to your distribution regardless of whether you specify the `Domain` attribute. For more information, see Alternate domain names (CNAMEs) in the topic All distribution settings reference.

Example 1

You can use the `Set-Cookie` headers for one signed cookie when you're using the domain name that is associated with your distribution in the URLs for your files.

```
Set-Cookie: CloudFront-
Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cDovL2QxMTExMTFhYmNkZWY4LmNsb3VkZnJvbnQubmV0L2dh
 Domain=d111111abcdef8.cloudfront.net; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Signature=dtKhpJ3aUYxqDIwepczPiDb9NXQ_;
 Domain=d111111abcdef8.cloudfront.net; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Key-Pair-Id=K2JCJMDEHXQW5F;
 Domain=d111111abcdef8.cloudfront.net; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Hash-Algorithm=SHA256; Domain=d111111abcdef8.cloudfront.net;
 Path=/; Secure; HttpOnly
```

Example 2

You can use the `Set-Cookie` headers for one signed cookie when you're using an alternate domain name (example.org) in the URLs for your files.

```
Set-Cookie: CloudFront-
Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cDovL2QxMTExMTFhYmNkZWY4LmNsb3VkZnJvbnQubmV0L2dh
 Domain=example.org; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Signature=dtKhpJ3aUYxqDIwepczPiDb9NXQ_; Domain=example.org;
 Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Key-Pair-Id=K2JCJMDEHXQW5F; Domain=example.org; Path=/; Secure;
 HttpOnly
Set-Cookie: CloudFront-Hash-Algorithm=SHA256; Domain=example.org; Path=/; Secure;
 HttpOnly
```

Example 3

You can use the `Set-Cookie` header pairs for a signed request when you're using the domain name that is associated with your distribution in the URLs for your files.

```
Set-Cookie: CloudFront-
Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cDovL2QxMTExMTFhYmNkZWY4LmNsb3VkZnJvbnQubmV0L2dh
 Domain=d111111abcdef8.cloudfront.net; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Signature=dtKhpJ3aUYxqDIwepczPiDb9NXQ_;
 Domain=d111111abcdef8.cloudfront.net; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Key-Pair-Id=K2JCJMDEHXQW5F;
 Domain=d111111abcdef8.cloudfront.net; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Hash-Algorithm=SHA256; Domain=d111111abcdef8.cloudfront.net;
 Path=/; Secure; HttpOnly
```

Example 4

You can use the `Set-Cookie` header pairs for one signed request when you're using an alternate domain name (example.org) that is associated with your distribution in the URLs for your files.

```
Set-Cookie: CloudFront-
Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc291cmNlIjoiaHR0cDovL2QxMTExMTFhYmNkZWY4LmNsb3VkZnJvbnQubmV0L2dh
 Domain=example.org; Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Signature=dtKhpJ3aUYxqDIwepczPiDb9NXQ_; Domain=example.org;
 Path=/; Secure; HttpOnly
Set-Cookie: CloudFront-Key-Pair-Id=K2JCJMDEHXQW5F; Domain=example.org; Path=/; Secure;
 HttpOnly
Set-Cookie: CloudFront-Hash-Algorithm=SHA256; Domain=example.org; Path=/; Secure;
 HttpOnly
```

Create a policy statement for a signed cookie that uses a custom policy

To create a policy statement for a custom policy, complete the following steps. For several example policy statements that control access to files in a variety of ways, see Example policy statements for a signed cookie that uses a custom policy.

To create the policy statement for a signed cookie that uses a custom policy

1. Construct the policy statement using the following JSON format.

   ```
   {
   ```

```
    "Statement": [
        {
            "Resource": "URL of the file",
            "Condition": {
                "DateLessThan": {
                    "AWS:EpochTime":required ending date and time in Unix time
 format and UTC
                },
                "DateGreaterThan": {
                    "AWS:EpochTime":optional beginning date and time in Unix time
 format and UTC
                },
                "IpAddress": {
                    "AWS:SourceIp": "optional IP address"
                }
            }
        }
    ]
}
```

Note the following:

- You can include only one statement.
- Use UTF-8 character encoding.
- Include all punctuation and parameter names exactly as specified. Abbreviations for parameter names are not accepted.
- The order of the parameters in the `Condition` section doesn't matter.
- For information about the values for `Resource`, `DateLessThan`, `DateGreaterThan`, and `IpAddress`, see Values that you specify in the policy statement for a custom policy for signed cookies.

2. Remove all empty spaces (including tabs and newline characters) from the policy statement. You might have to include escape characters in the string in application code.
3. Base64-encode the policy statement using MIME base64 encoding. For more information, see Section 6.8, Base64 Content-Transfer-Encoding in *RFC 2045, MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*.
4. Replace characters that are invalid in a URL query string with characters that are valid. The following table lists invalid and valid characters.

Replace these invalid characters	With these valid characters
+	- (hyphen)
=	_ (underscore)
/	~ (tilde)

5. Include the resulting value in your `Set-Cookie` header after `CloudFront-Policy=`.
6. Create a signature for the `Set-Cookie` header for `CloudFront-Signature` by hashing, signing, and base64-encoding the policy statement. For more information, see Create a signature for a signed cookie that uses a custom policy.

Values that you specify in the policy statement for a custom policy for signed cookies

When you create a policy statement for a custom policy, you specify the following values.

Resource

The base URL including your query strings, if any:

```
https://d111111abcdef8.cloudfront.net/images/horizon.jpg?
size=large&license=yes
```

> **Important**
>
> If you omit the `Resource` parameter, users can access all of the files associated with any distribution that is associated with the key pair that you use to create the signed URL.

You can specify only one value for `Resource`.

Note the following:

- **Protocol** – The value must begin with `http://` or `https://`.

- **Query string parameters** – If you have no query string parameters, omit the question mark.
- **Wildcards** – You can use the wildcard character that matches zero or more characters (*) or the wild-card character that matches exactly one character (?) anywhere in the string. For example, the value:

  `https://d111111abcdef8.cloudfront.net/*game_download.zip*`

  would include (for example) the following files:

  - `https://d111111abcdef8.cloudfront.net/game_download.zip`
  - `https://d111111abcdef8.cloudfront.net/example_game_download.zip?license=yes`
  - `https://d111111abcdef8.cloudfront.net/test_game_download.zip?license=temp`

  A percent-encoded character in the request URL (such as `%3B` for `;`, `%3F` for `?`, or `%23` for #) is treated as data, not as a URL delimiter. For example, a wildcard can't match across a literal semicolon (`;`), but it can match across the encoded form (`%3B`). For more information, see Values that you specify in the policy statement for a signed URL that uses a custom policy.
- **Alternate domain names** – If you specify an alternate domain name (CNAME) in the URL, you must specify the alternate domain name when referencing the file in your webpage or application. Do not specify the Amazon S3 URL for the file.

DateLessThan

The expiration date and time for the URL in Unix time format (in seconds) and Coordinated Universal Time (UTC). Do not enclose the value in quotation marks.

For example, March 16, 2015 10:00 am UTC converts to 1426500000 in Unix time format.

For more information, see When CloudFront checks expiration date and time in a signed cookie.

DateGreaterThan (Optional)

An optional start date and time for the URL in Unix time format (in seconds) and Coordinated Universal Time (UTC). Users are not allowed to access the file on or before the specified date and time. Do not enclose the value in quotation marks.

IpAddress (Optional)

The IP address of the client making the GET request. Note the following:

- To allow any IP address to access the file, omit the `IpAddress` parameter.
- You can specify either one IP address or one IP address range. For example, you can't set the policy to allow access if the client's IP address is in one of two separate ranges.
- To allow access from a single IP address, you specify:

  `"IPv4 IP address/32"`
- You must specify IP address ranges in standard IPv4 CIDR format (for example, `192.0.2.0/24`). For more information, go to *RFC 4632, Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*, https://tools.ietf.org/html/rfc4632.

  > **Important**
  >
  > IP addresses in IPv6 format, such as 2001:0db8:85a3::8a2e:0370:7334, are not supported.

  If you're using a custom policy that includes `IpAddress`, do not enable IPv6 for the distribution. If you want to restrict access to some content by IP address and support IPv6 requests for other content, you can create two distributions. For more information, see Enable IPv6 (viewer requests) in the topic All distribution settings reference.

Example policy statements for a signed cookie that uses a custom policy

The following example policy statements show how to control access to a specific file, all of the files in a directory, or all of the files associated with a key pair ID. The examples also show how to control access from an individual IP address or a range of IP addresses, and how to prevent users from using the signed cookie after a specified date and time.

If you copy and paste any of these examples, remove any empty spaces (including tabs and newline characters), replace the values with your own values, and include a newline character after the closing brace (}).

For more information, see Values that you specify in the policy statement for a custom policy for signed cookies.

Topics

- Example policy statement: Access one file from a range of IP addresses

- Example policy statement: Access all files in a directory from a range of IP addresses
- Example policy statement: Access all files associated with a key pair ID from one IP address

Example policy statement: Access one file from a range of IP addresses

The following example custom policy in a signed cookie specifies that a user can access the file `https://d111111abcdef8.cloudfront.net/game_download.zip` from IP addresses in the range `192.0.2.0/24` until January 1, 2023 10:00 am UTC:

```
{
    "Statement": [
        {
            "Resource": "https://d111111abcdef8.cloudfront.net/game_download.zip",
            "Condition": {
                "IpAddress": {
                    "AWS:SourceIp": "192.0.2.0/24"
                },
                "DateLessThan": {
                    "AWS:EpochTime": 1767290400
                }
            }
        }
    ]
}
```

Example policy statement: Access all files in a directory from a range of IP addresses

With the following example custom policy, you can create signed cookies for any file in the `training` directory, as indicated by the * wildcard character in the `Resource` parameter. Users can access the file from an IP address in the range `192.0.2.0/24` until January 1, 2013 10:00 am UTC:

```
{
    "Statement": [
        {
            "Resource": "https://d111111abcdef8.cloudfront.net/training/*",
            "Condition": {
                "IpAddress": {
                    "AWS:SourceIp": "192.0.2.0/24"
                },
                "DateLessThan": {
```

```
                    "AWS:EpochTime": 1767290400
                }
            }
        }
    ]
}
```

Each signed cookie in which you use this policy includes a base URL that identifies a specific file, for example:

`https://d111111abcdef8.cloudfront.net/training/orientation.pdf`

Example policy statement: Access all files associated with a key pair ID from one IP address

With the following sample custom policy, you can set signed cookies for any file associated with any distribution, as indicated by the * wildcard character in the `Resource` parameter. The user must use the IP address `192.0.2.10/32`. (The value `192.0.2.10/32` in CIDR notation refers to a single IP address, `192.0.2.10`.) The files are available only from January 1, 2013 10:00 am UTC until January 2, 2013 10:00 am UTC:

```
{
    "Statement": [
        {
            "Resource": "https://*",
            "Condition": {
                "IpAddress": {
                    "AWS:SourceIp": "192.0.2.10/32"
                },
                "DateGreaterThan": {
                    "AWS:EpochTime": 1767290400
                },
                "DateLessThan": {
                    "AWS:EpochTime": 1767376800
                }
            }
        }
    ]
}
```

Each signed cookie in which you use this policy includes a base URL that identifies a specific file in a specific CloudFront distribution, for example:

```
https://d111111abcdef8.cloudfront.net/training/orientation.pdf
```

The signed cookie also includes a key pair ID, which must be associated with a trusted key group in the distribution (d111111abcdef8.cloudfront.net) that you specify in the base URL.

Create a signature for a signed cookie that uses a custom policy

The signature for a signed cookie that uses a custom policy is a hashed, signed, and base64-encoded version of the policy statement.

For additional information and examples of how to hash, sign, and encode the policy statement, see:

- Linux commands and OpenSSL for base64 encoding and encryption
- Code examples for creating a signature for a signed URL

> **Note**
>
> The linked examples use SHA-1 by default. To use SHA-256 instead, replace `sha1` with `sha256` in the OpenSSL commands and include the `CloudFront-Hash-Algorithm` cookie with a value of `SHA256`.

To create a signature for a signed cookie by using a custom policy

1. Use the SHA-1 or SHA-256 hash function and RSA to hash and sign the JSON policy statement that you created in the procedure To create the policy statement for a signed URL that uses a custom policy. Use the version of the policy statement that no longer includes empty spaces but that has not yet been base64-encoded.

   If you use SHA-256, you must include the `CloudFront-Hash-Algorithm` cookie with a value of `SHA256`.

   For the private key that is required by the hash function, use a private key whose public key is in an active trusted key group for the distribution.

> **Note**
>
> The method that you use to hash and sign the policy statement depends on your programming language and platform. For sample code, see Code examples for creating a signature for a signed URL.

2. Remove empty spaces (including tabs and newline characters) from the hashed and signed string.
3. Base64-encode the string using MIME base64 encoding. For more information, see Section 6.8, Base64 Content-Transfer-Encoding in *RFC 2045, MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies*.
4. Replace characters that are invalid in a URL query string with characters that are valid. The following table lists invalid and valid characters.

Replace these invalid characters	With these valid characters
+	- (hyphen)
=	_ (underscore)
/	~ (tilde)

5. Include the resulting value in the `Set-Cookie` header for the `CloudFront-Signature=` name-value pair, and return to To set a signed cookie using a custom policy to add the `Set-Cookie` header for `CloudFront-Key-Pair-Id`.

Create signed cookies using PHP

The following code example is similar to the example in the Create a URL signature using PHP in that it creates a link to a video. However, instead of signing the URL in the code, this example signs the cookies with the `create_signed_cookies()` function. The client-side player uses the cookies to authenticate each request to the CloudFront distribution.

This approach is useful to stream content, such as HTTP Live Streaming (HLS) or Dynamic Adaptive Streaming over HTTP (DASH), where the client needs to make multiple requests to retrieve

the manifest, segments, and related playback assets. By using signed cookies, the client can authenticate each request without needing to generate a new signed URL for each segment.

Note

- Creating a URL signature is just one part of the process of serving private content using signed cookies. For more information, see Use signed cookies.

Topics

- Create the RSA SHA-1 or SHA-256 signature
- Create the signed cookies
- Full code

The following sections break down the code example into individual parts. You can find the complete code sample below.

Create the RSA SHA-1 or SHA-256 signature

This code example does the following:

1. The function `rsa_sha1_sign` hashes and signs the policy statement using SHA-1. To use SHA-256 instead, use the rsa_sha256_sign function shown below. The arguments required are a policy statement and the private key that corresponds with a public key that's in a trusted key group for your distribution.
2. Next, the `url_safe_base64_encode` function creates a URL-safe version of the signature.

```
function rsa_sha1_sign($policy, $private_key_filename) {
    $signature = "";
    $fp = fopen($private_key_filename, "r");
    $priv_key = fread($fp, 8192);
    fclose($fp);
    $pkeyid = openssl_get_privatekey($priv_key);
    openssl_sign($policy, $signature, $pkeyid);
    openssl_free_key($pkeyid);
    return $signature;
}
```

```
function url_safe_base64_encode($value) {
    $encoded = base64_encode($value);
    return str_replace(
        array('+', '=', '/'),
        array('-', '_', '~'),
        $encoded);
}
```

The following function uses SHA-256 instead of SHA-1:

```
function rsa_sha256_sign($policy, $private_key_filename) {
    $signature = "";
    $fp = fopen($private_key_filename, "r");
    $priv_key = fread($fp, 8192);
    fclose($fp);
    $pkeyid = openssl_get_privatekey($priv_key);
    openssl_sign($policy, $signature, $pkeyid, OPENSSL_ALGO_SHA256);
    openssl_free_key($pkeyid);
    return $signature;
}
```

The `rsa_sha256_sign` function is the same as `rsa_sha1_sign`, except that it passes `OPENSSL_ALGO_SHA256` to `openssl_sign`. When you use SHA-256, include the `CloudFront-Hash-Algorithm` cookie with a value of `SHA256`.

Create the signed cookies

The following code constructs a creates the signed cookies, using the following cookie attributes: `CloudFront-Expires`, `CloudFront-Signature`, `CloudFront-Key-Pair-Id` and `CloudFront-Hash-Algorithm`. The code uses a custom policy.

```
function create_signed_cookies($resource, $private_key_filename, $key_pair_id,
 $expires, $client_ip = null, $hash_algorithm = 'SHA1') {
   $policy = array(
       'Statement' => array(
           array(
               'Resource' => $resource,
               'Condition' => array(
                   'DateLessThan' => array('AWS:EpochTime' => $expires)
               )
           )
```

```
        )
    );

    if ($client_ip) {
        $policy['Statement'][0]['Condition']['IpAddress'] = array('AWS:SourceIp' =>
 $client_ip . '/32');
    }

    $policy = json_encode($policy);
    $encoded_policy = url_safe_base64_encode($policy);
    if ($hash_algorithm === 'SHA256') {
        $signature = rsa_sha256_sign($policy, $private_key_filename);
    } else {
        $signature = rsa_sha1_sign($policy, $private_key_filename);
    }
    $encoded_signature = url_safe_base64_encode($signature);

    $cookies = array(
        'CloudFront-Policy' => $encoded_policy,
        'CloudFront-Signature' => $encoded_signature,
        'CloudFront-Key-Pair-Id' => $key_pair_id
    );

    if ($hash_algorithm === 'SHA256') {
        $cookies['CloudFront-Hash-Algorithm'] = 'SHA256';
    }

    return $cookies;
}
```

For more information, see Set signed cookies using a custom policy.

Full code

The following example code provides a complete demonstration of creating CloudFront signed cookies with PHP. You can download the full example from the demo-php.zip file.

In the following example, you can modify the `$policy` `Condition` element to allow both IPv4 and IPv6 address ranges. For an example, see Using IPv6 addresses in IAM policies in the *Amazon Simple Storage Service User Guide*.

```
<?php
```

```
function rsa_sha1_sign($policy, $private_key_filename) {
    $signature = "";
    $fp = fopen($private_key_filename, "r");
    $priv_key = fread($fp, 8192);
    fclose($fp);
    $pkeyid = openssl_get_privatekey($priv_key);
    openssl_sign($policy, $signature, $pkeyid);
    openssl_free_key($pkeyid);
    return $signature;
}

function url_safe_base64_encode($value) {
    $encoded = base64_encode($value);
    return str_replace(
        array('+', '=', '/'),
        array('-', '_', '~'),
        $encoded);
}

function rsa_sha256_sign($policy, $private_key_filename) {
    $signature = "";
    $fp = fopen($private_key_filename, "r");
    $priv_key = fread($fp, 8192);
    fclose($fp);
    $pkeyid = openssl_get_privatekey($priv_key);
    openssl_sign($policy, $signature, $pkeyid, OPENSSL_ALGO_SHA256);
    openssl_free_key($pkeyid);
    return $signature;
}

function create_signed_cookies($resource, $private_key_filename, $key_pair_id,
 $expires, $client_ip = null, $hash_algorithm = 'SHA1') {
    $policy = array(
        'Statement' => array(
            array(
                'Resource' => $resource,
                'Condition' => array(
                    'DateLessThan' => array('AWS:EpochTime' => $expires)
                )
            )
        )
    );

    if ($client_ip) {
```

```
        $policy['Statement'][0]['Condition']['IpAddress'] = array('AWS:SourceIp' =>
 $client_ip . '/32');
    }

    $policy = json_encode($policy);
    $encoded_policy = url_safe_base64_encode($policy);
    if ($hash_algorithm === 'SHA256') {
        $signature = rsa_sha256_sign($policy, $private_key_filename);
    } else {
        $signature = rsa_sha1_sign($policy, $private_key_filename);
    }
    $encoded_signature = url_safe_base64_encode($signature);

    $cookies = array(
        'CloudFront-Policy' => $encoded_policy,
        'CloudFront-Signature' => $encoded_signature,
        'CloudFront-Key-Pair-Id' => $key_pair_id
    );

    if ($hash_algorithm === 'SHA256') {
        $cookies['CloudFront-Hash-Algorithm'] = 'SHA256';
    }

    return $cookies;
}



$private_key_filename = '/home/test/secure/example-priv-key.pem';
$key_pair_id = 'K2JCJMDEHXQW5F';
$base_url = 'https://d1234.cloudfront.net';

$expires = time() + 3600; // 1 hour from now

// Get the viewer real IP from the x-forwarded-for header as $_SERVER['REMOTE_ADDR']
 will return viewer facing IP. An alternative option is to use CloudFront-Viewer-
Address header. Note that this header is a trusted CloudFront immutable header. Example
 format: IP:PORT ("CloudFront-Viewer-Address": "1.2.3.4:12345")
$client_ip = $_SERVER['HTTP_X_FORWARDED_FOR'];


// For HLS manifest and segments (using wildcard)
$hls_resource = $base_url . '/sign/*';
```

```
$signed_cookies = create_signed_cookies($hls_resource, $private_key_filename,
 $key_pair_id, $expires, $client_ip, 'SHA256');

// Set the cookies
$cookie_domain = parse_url($base_url, PHP_URL_HOST);
foreach ($signed_cookies as $name => $value) {
    setcookie($name, $value, $expires, '/', $cookie_domain, true, true);
}

?>

<!DOCTYPE html>
<html>
<head>
    <title>CloudFront Signed HLS Stream with Cookies</title>
</head>
<body>
    <h1>Amazon CloudFront Signed HLS Stream with Cookies</h1>
    <h2>Expires at <?php echo gmdate('Y-m-d H:i:s T', $expires); ?> only viewable by IP
 <?php echo $client_ip; ?></h2>

    <div id='hls-video'>
        <video id="video" width="640" height="360" controls></video>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/hls.js@latest"></script>
    <script>
        var video = document.getElementById('video');
        var manifestUrl = '<?php echo $base_url; ?>/sign/manifest.m3u8';

        if (Hls.isSupported()) {
            var hls = new Hls();
            hls.loadSource(manifestUrl);
            hls.attachMedia(video);
        }
        else if (video.canPlayType('application/vnd.apple.mpegurl')) {
            video.src = manifestUrl;
        }
    </script>
</body>
</html>
```

Instead of using signed cookies, you can use signed URLs. For more information, see Create a URL signature using PHP.

Linux commands and OpenSSL for base64 encoding and encryption

You can use the following Linux command-line command and OpenSSL to hash and sign the policy statement, base64-encode the signature, and replace characters that are not valid in URL query string parameters with characters that are valid.

For information about OpenSSL, go to https://www.openssl.org.

SHA-1 (default):

```
cat policy | tr -d "\n" | tr -d " \t\n\r" | openssl sha1 -sign private_key.pem |
 openssl base64 -A | tr -- '+=/' '-_~'
```

SHA-256:

```
cat policy | tr -d "\n" | tr -d " \t\n\r" | openssl sha256 -sign private_key.pem |
 openssl base64 -A | tr -- '+=/' '-_~'
```

In the preceding command:

- `cat` reads the `policy` file.
- `tr -d "\n" | tr -d " \t\n\r"` removes the empty spaces and newline character that were added by `cat`.
- OpenSSL hashes the file using SHA-1 (or SHA-256) and signs it using the private key file `private_key.pem`. The private key signature can be either RSA 2048 or ECDSA 256. If you use SHA-256, include the `Hash-Algorithm=SHA256` query parameter in the signed URL, or the `CloudFront-Hash-Algorithm=SHA256` cookie for signed cookies.
- OpenSSL base64-encodes the hashed and signed policy statement.
- `tr` replaces characters that are not valid in URL query string parameters with characters that are valid.

For more code examples that demonstrate creating a signature, see Code examples for creating a signature for a signed URL.

Code examples for creating a signature for a signed URL

This section includes downloadable application examples that demonstrate how to create signatures for signed URLs. Examples are available in Perl, PHP, C#, and Java. You can use any of the examples to create signed URLs. The Perl script runs on Linux and macOS platforms. The PHP example will work on any server that runs PHP. The C# example uses the .NET Framework.

The examples in this section use SHA-1 to hash and sign the policy statement. You can also use SHA-256. To use SHA-256, update the hash algorithm in the signing function (for example, replace `sha1` with `sha256` in OpenSSL calls, or use the equivalent SHA-256 constant in your language's cryptographic library). When you use SHA-256, include the `Hash-Algorithm=SHA256` query parameter in the signed URL.

For example code in JavaScript (Node.js), see Creating Amazon CloudFront Signed URLs in Node.js on the AWS Developer Blog.

For example code in Python, see Generate a signed URL for Amazon CloudFront in the *AWS SDK for Python (Boto3) API Reference* and this example code in the Boto3 GitHub repository.

Topics

- Create a URL signature using Perl
- Create a URL signature using PHP
- Create a URL signature using C# and the .NET Framework
- Create a URL signature using Java

Create a URL signature using Perl

This section includes a Perl script for Linux/Mac platforms that you can use to create the signature for private content. To create the signature, run the script with command line arguments that specify the CloudFront URL, the path to the private key of the signer, the key ID, and an expiration date for the URL. The tool can also decode signed URLs.

Notes

- Creating a URL signature is just one part of the process of serving private content using a signed URL. For more information about the end-to-end process, see Use signed URLs.

- In the signing command, note that `sha1` can be replaced with `sha256` in the `openssl dgst` call.

Topics

- Source for the Perl script to create a signed URL

Source for the Perl script to create a signed URL

The following Perl source code can be used to create a signed URL for CloudFront. Comments in the code include information about the command line switches and the features of the tool.

```
#!/usr/bin/perl -w

# Copyright 2008 Amazon Technologies, Inc.  Licensed under the Apache License, Version
 2.0 (the "License");
# you may not use this file except in compliance with the License. You may obtain a
 copy of the License at:
#
# https://aws.amazon.com/apache2.0
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 KIND, either express or implied.
# See the License for the specific language governing permissions and limitations under
 the License.

=head1 cfsign.pl

cfsign.pl - A tool to generate and verify Amazon CloudFront signed URLs

=head1 SYNOPSIS

This script uses an existing RSA key pair to sign and verify Amazon CloudFront signed
 URLs

View the script source for details as to which CPAN packages are required beforehand.

For help, try:

cfsign.pl --help
```

```
URL signing examples:

cfsign.pl --action encode --url https://images.my-website.com/gallery1.zip --policy
 sample_policy.json --private-key privkey.pem --key-pair-id mykey

cfsign.pl --action encode --url https://images.my-website.com/gallery1.zip --expires
 1257439868 --private-key privkey.pem --key-pair-id mykey

URL decode example:

cfsign.pl --action decode --url "http://mydist.cloudfront.net/?Signature=AGO-
PgxkYo99MkJFHvjfGXjG1QDEXeaDb4Qtzmy85wqyJjK7eKojQWa4BCRcow__&Policy=eyJTdGF0ZW1lbnQiOlt7IlJlc29
Pair-Id=mykey"


To generate an RSA key pair, you can use openssl and the following commands:

# Generate a 2048 bit key pair
openssl genrsa -out private-key.pem 2048
openssl rsa -in private-key.pem -pubout -out public-key.pem


=head1 OPTIONS

=over 8

=item B<--help>

Print a help message and exits.

=item B<--action> [action]

The action to execute.  action can be one of:

  encode - Generate a signed URL (using a canned policy or a user policy)
  decode - Decode a signed URL

=item B<--url>

The URL to en/decode

=item B<--stream>

The stream to en/decode
```

```
=item B<--private-key>

The path to your private key.

=item B<--key-pair-id>

The key pair identifier.

=item B<--policy>

The CloudFront policy document.

=item B<--expires>

The Unix epoch time when the URL is to expire. If both this option and
the --policy option are specified, --policy will be used. Otherwise, this
option alone will use a canned policy.

=back

=cut

use strict;
use warnings;

# you might need to use CPAN to get these modules.
# run perl -MCPAN -e "install <module>" to get them.
# The openssl command line will also need to be in your $PATH.
use File::Temp qw/tempfile/;
use File::Slurp;
use Getopt::Long;
use IPC::Open2;
use MIME::Base64 qw(encode_base64 decode_base64);
use Pod::Usage;
use URI;

my $CANNED_POLICY
    = '{"Statement":[{"Resource":"<RESOURCE>","Condition":{"DateLessThan":
{"AWS:EpochTime":<EXPIRES>}}}]}';

my $POLICY_PARAM      = "Policy";
my $EXPIRES_PARAM     = "Expires";
my $SIGNATURE_PARAM   = "Signature";
```

```
my $KEY_PAIR_ID_PARAM = "Key-Pair-Id";

my $verbose = 0;
my $policy_filename = "";
my $expires_epoch = 0;
my $action = "";
my $help = 0;
my $key_pair_id = "";
my $url = "";
my $stream = "";
my $private_key_filename = "";

my $result = GetOptions("action=s"      => \$action,
                        "policy=s"      => \$policy_filename,
                        "expires=i"     => \$expires_epoch,
                        "private-key=s" => \$private_key_filename,
                        "key-pair-id=s" => \$key_pair_id,
                        "verbose"       => \$verbose,
                        "help"          => \$help,
                        "url=s"         => \$url,
                        "stream=s"      => \$stream,
                    );

if ($help or !$result) {
    pod2usage(1);
    exit;
}

if ($url eq "" and $stream eq "") {
    print STDERR "Must include a stream or a URL to encode or decode with the --stream
 or --url option\n";
    exit;
}

if ($url ne "" and $stream ne "") {
    print STDERR "Only one of --url and --stream may be specified\n";
    exit;
}

if ($url ne "" and !is_url_valid($url)) {
    exit;
}

if ($stream ne "") {
```

```
    exit unless is_stream_valid($stream);

    # The signing mechanism is identical, so from here on just pretend we're
    # dealing with a URL
    $url = $stream;
}

if ($action eq "encode") {
    # The encode action will generate a private content URL given a base URL,
    # a policy file (or an expires timestamp) and a key pair id parameter
    my $private_key;
    my $public_key;
    my $public_key_file;

    my $policy;
    if ($policy_filename eq "") {
        if ($expires_epoch == 0) {
            print STDERR "Must include policy filename with --policy argument or an
 expires" .
                         "time using --expires\n";
        }

        $policy = $CANNED_POLICY;
        $policy =~ s/<EXPIRES>/$expires_epoch/g;
        $policy =~ s/<RESOURCE>/$url/g;
    } else {
        if (! -e $policy_filename) {
            print STDERR "Policy file $policy_filename does not exist\n";
            exit;
        }
        $expires_epoch = 0; # ignore if set
        $policy = read_file($policy_filename);
    }

    if ($private_key_filename eq "") {
        print STDERR "You must specify the path to your private key file with --
private-key\n";
        exit;
    }

    if (! -e $private_key_filename) {
        print STDERR "Private key file $private_key_filename does not exist\n";
        exit;
    }
```

```
    if ($key_pair_id eq "") {
        print STDERR "You must specify a key pair id with --key-pair-id\n";
        exit;
    }

    my $encoded_policy = url_safe_base64_encode($policy);
    my $signature = rsa_sha1_sign($policy, $private_key_filename);
    my $encoded_signature = url_safe_base64_encode($signature);

    my $generated_url = create_url($url, $encoded_policy, $encoded_signature,
 $key_pair_id, $expires_epoch);


    if ($stream ne "") {
        print "Encoded stream (for use within a swf):\n" . $generated_url . "\n";
        print "Encoded and escaped stream (for use on a webpage):\n" .
 escape_url_for_webpage($generated_url) . "\n";
    } else {
        print "Encoded URL:\n" . $generated_url . "\n";
    }
} elsif ($action eq "decode") {
    my $decoded = decode_url($url);
    if (!$decoded) {
        print STDERR "Improperly formed URL\n";
        exit;
    }

    print_decoded_url($decoded);
} else {
    # No action specified, print help.  But only if this is run as a program (caller
 will be empty)
    pod2usage(1) unless caller();
}

# Decode a private content URL into its component parts
sub decode_url {
    my $url = shift;

    if ($url =~ /(.*)\?(.*)/) {
        my $base_url = $1;
        my $params = $2;

        my @unparsed_params = split(/&/, $params);
```

```
        my %params = ();
        foreach my $param (@unparsed_params) {
            my ($key, $val) = split(/=/, $param);
            $params{$key} = $val;
        }

        my $encoded_signature = "";
        if (exists $params{$SIGNATURE_PARAM}) {
            $encoded_signature = $params{"Signature"};
        } else {
            print STDERR "Missing Signature URL parameter\n";
            return 0;
        }

        my $encoded_policy = "";
        if (exists $params{$POLICY_PARAM}) {
            $encoded_policy = $params{$POLICY_PARAM};
        } else {
            if (!exists $params{$EXPIRES_PARAM}) {
                print STDERR "Either the Policy or Expires URL parameter needs to be
specified\n";
                return 0;
            }

            my $expires = $params{$EXPIRES_PARAM};

            my $policy = $CANNED_POLICY;
            $policy =~ s/<EXPIRES>/$expires/g;

            my $url_without_cf_params = $url;
            $url_without_cf_params =~ s/$SIGNATURE_PARAM=[^&]*&?//g;
            $url_without_cf_params =~ s/$POLICY_PARAM=[^&]*&?//g;
            $url_without_cf_params =~ s/$EXPIRES_PARAM=[^&]*&?//g;
            $url_without_cf_params =~ s/$KEY_PAIR_ID_PARAM=[^&]*&?//g;

            if ($url_without_cf_params =~ /(.*)\?$/) {
                $url_without_cf_params = $1;
            }

            $policy =~ s/<RESOURCE>/$url_without_cf_params/g;

            $encoded_policy = url_safe_base64_encode($policy);
        }
```

```
        my $key = "";
        if (exists $params{$KEY_PAIR_ID_PARAM}) {
            $key = $params{$KEY_PAIR_ID_PARAM};
        } else {
            print STDERR "Missing $KEY_PAIR_ID_PARAM parameter\n";
            return 0;
        }

        my $policy = url_safe_base64_decode($encoded_policy);

        my %ret = ();
        $ret{"base_url"} = $base_url;
        $ret{"policy"} = $policy;
        $ret{"key"} = $key;

        return \%ret;
    } else {
        return 0;
    }
}

# Print a decoded URL out
sub print_decoded_url {
    my $decoded = shift;

    print "Base URL: \n" . $decoded->{"base_url"} . "\n";
    print "Policy: \n" . $decoded->{"policy"} . "\n";
    print "Key: \n" . $decoded->{"key"} . "\n";
}

# Encode a string with base 64 encoding and replace some invalid URL characters
sub url_safe_base64_encode {
    my ($value) = @_;

    my $result = encode_base64($value);
    $result =~ tr|+=/|-_~|;

    return $result;
}

# Decode a string with base 64 encoding.  URL-decode the string first
# followed by reversing any special character ("+=/") translation.
sub url_safe_base64_decode {
    my ($value) = @_;
```

```
    $value =~ s/%([0-9A-Fa-f]{2})/chr(hex($1))/eg;
    $value =~ tr|-_~|+=/|;

    my $result = decode_base64($value);

    return $result;
}

# Create a private content URL
sub create_url {
    my ($path, $policy, $signature, $key_pair_id, $expires) = @_;

    my $result;
    my $separator = $path =~ /\?/ ? '&' : '?';
    if ($expires) {
        $result = "$path$separator$EXPIRES_PARAM=$expires&$SIGNATURE_PARAM=$signature&
$KEY_PAIR_ID_PARAM=$key_pair_id";
    } else {
        $result = "$path$separator$POLICY_PARAM=$policy&$SIGNATURE_PARAM=$signature&
$KEY_PAIR_ID_PARAM=$key_pair_id";
    }
    $result =~ s/\n//g;

    return $result;
}

# Sign a document with given private key file.
# The first argument is the document to sign
# The second argument is the name of the private key file
sub rsa_sha1_sign {
    my ($to_sign, $pvkFile) = @_;
    print "openssl sha1 -sign $pvkFile $to_sign\n";

    return write_to_program($pvkFile, $to_sign);
}

# Helper function to write data to a program
sub write_to_program {
my ($keyfile, $data) = @_;
unlink "temp_policy.dat" if (-e "temp_policy.dat");
unlink "temp_sign.dat" if (-e "temp_sign.dat");

write_file("temp_policy.dat", $data);
```

```
system("openssl dgst -sha1 -sign \"$keyfile\" -out temp_sign.dat temp_policy.dat");

my $output = read_file("temp_sign.dat");

    return $output;
}

# Read a file into a string and return the string
sub read_file {
    my ($file) = @_;

    open(INFILE, "<$file") or die("Failed to open $file: $!");
    my $str = join('', <INFILE>);
    close INFILE;

    return $str;
}

sub is_url_valid {
    my ($url) = @_;

    # HTTP distributions start with http[s]:// and are the correct thing to sign
    if ($url =~ /^https?:\/\//) {
        return 1;
    } else {
        print STDERR "CloudFront requires absolute URLs for HTTP distributions\n";
        return 0;
    }
}

sub is_stream_valid {
    my ($stream) = @_;

    if ($stream =~ /^rtmp:\/\// or $stream =~ /^\/?cfx\/st/) {
        print STDERR "Streaming distributions require that only the stream name is
 signed.\n";
        print STDERR "The stream name is everything after, but not including, cfx/st/
\n";
        return 0;
    } else {
        return 1;
    }
}
```

```
# flash requires that the query parameters in the stream name are url
# encoded when passed in through javascript, etc.  This sub handles the minimal
# required url encoding.
sub escape_url_for_webpage {
    my ($url) = @_;

    $url =~ s/\?/%3F/g;
    $url =~ s/=/%3D/g;
    $url =~ s/&/%26/g;

    return $url;
}

1;
```

Create a URL signature using PHP

Any web server that runs PHP can use this PHP example code to create policy statements and signatures for private CloudFront distributions. The full example creates a functioning webpage with signed URL links that play a video stream using CloudFront streaming. You can download the full example from the demo-php.zip file.

Notes

- Creating a URL signature is just one part of the process of serving private content using a signed URL. For more information about the entire process, see Use signed URLs.
- You can also create signed URLs by using the `UrlSigner` class in the AWS SDK for PHP. For more information, see Class UrlSigner in the *AWS SDK for PHP API Reference*.
- In the `openssl_sign` call, note that passing `OPENSSL_ALGO_SHA256` as the fourth argument switches to SHA-256. (See also the Create signed cookies using PHP for a full example.)

Topics

- Create the RSA SHA-1 signature
- Create a canned policy
- Create a custom policy

- Full code example

The following sections break down the code example into individual parts. You can find the Full code example below.

Create the RSA SHA-1 signature

This code example does the following:

- The function `rsa_sha1_sign` hashes and signs the policy statement. The arguments required are a policy statement and the private key that corresponds with a public key that's in a trusted key group for your distribution.
- Next, the `url_safe_base64_encode` function creates a URL-safe version of the signature.

```
function rsa_sha1_sign($policy, $private_key_filename) {
    $signature = "";

    // load the private key
    $fp = fopen($private_key_filename, "r");
    $priv_key = fread($fp, 8192);
    fclose($fp);
    $pkeyid = openssl_get_privatekey($priv_key);

    // compute signature
    openssl_sign($policy, $signature, $pkeyid);

    // free the key from memory
    openssl_free_key($pkeyid);

    return $signature;
}

function url_safe_base64_encode($value) {
    $encoded = base64_encode($value);
    // replace unsafe characters +, = and / with
    // the safe characters -, _ and ~
    return str_replace(
        array('+', '=', '/'),
        array('-', '_', '~'),
        $encoded);
```

```
}
```

The following code snippet uses the functions `get_canned_policy_stream_name()` and `get_custom_policy_stream_name()` to create a canned and custom policy. CloudFront uses the policies to create the URL for streaming the video, including specifying the expiration time.

You can then use a canned policy or a custom policy to determine how to manage access to your content. For more information about which one to choose, see the Decide to use canned or custom policies for signed URLs section.

Create a canned policy

The following example code constructs a *canned* policy statement for the signature.

> **Note**
>
> The `$expires` variable is a date/time stamp that must be an integer, not a string.

```
function get_canned_policy_stream_name($video_path, $private_key_filename,
 $key_pair_id, $expires) {
    // this policy is well known by CloudFront, but you still need to sign it, since it
 contains your parameters
    $canned_policy = '{"Statement":[{"Resource":"' . $video_path . '","Condition":
{"DateLessThan":{"AWS:EpochTime":'. $expires . '}}}]}';
    // the policy contains characters that cannot be part of a URL, so we base64 encode
 it
    $encoded_policy = url_safe_base64_encode($canned_policy);
    // sign the original policy, not the encoded version
    $signature = rsa_sha1_sign($canned_policy, $private_key_filename);
    // make the signature safe to be included in a URL
    $encoded_signature = url_safe_base64_encode($signature);

    // combine the above into a stream name
    $stream_name = create_stream_name($video_path, null, $encoded_signature,
 $key_pair_id, $expires);
    // URL-encode the query string characters
    return $stream_name;
}
```

For more information about canned policies, see Create a signed URL using a canned policy.

Create a custom policy

The following example code constructs a *custom* policy statement for the signature.

```
function get_custom_policy_stream_name($video_path, $private_key_filename,
 $key_pair_id, $policy) {
    // the policy contains characters that cannot be part of a URL, so we base64 encode
 it
    $encoded_policy = url_safe_base64_encode($policy);
    // sign the original policy, not the encoded version
    $signature = rsa_sha1_sign($policy, $private_key_filename);
    // make the signature safe to be included in a URL
    $encoded_signature = url_safe_base64_encode($signature);

    // combine the above into a stream name
    $stream_name = create_stream_name($video_path, $encoded_policy, $encoded_signature,
 $key_pair_id, null);
    // URL-encode the query string characters
    return $stream_name;
}
```

For more information about custom policies, see Create a signed URL using a custom policy.

Full code example

The following example code provides a complete demonstration of creating CloudFront signed URLs with PHP. You can download the full example from the demo-php.zip file.

In the following example, you can modify the `$policy` `Condition` element to allow both IPv4 and IPv6 address ranges. For an example, see Using IPv6 addresses in IAM policies in the *Amazon Simple Storage Service User Guide*.

```
<?php

function rsa_sha1_sign($policy, $private_key_filename) {
    $signature = "";

    // load the private key
    $fp = fopen($private_key_filename, "r");
    $priv_key = fread($fp, 8192);
    fclose($fp);
    $pkeyid = openssl_get_privatekey($priv_key);
```

```
    // compute signature
    openssl_sign($policy, $signature, $pkeyid);

    // free the key from memory
    openssl_free_key($pkeyid);

    return $signature;
}

function url_safe_base64_encode($value) {
    $encoded = base64_encode($value);
    // replace unsafe characters +, = and / with the safe characters -, _ and ~
    return str_replace(
        array('+', '=', '/'),
        array('-', '_', '~'),
        $encoded);
}

function create_stream_name($stream, $policy, $signature, $key_pair_id, $expires) {
    $result = $stream;
    // if the stream already contains query parameters, attach the new query parameters
 to the end
    // otherwise, add the query parameters
    $separator = strpos($stream, '?') == FALSE ? '?' : '&';
    // the presence of an expires time means we're using a canned policy
    if($expires) {
        $result .= $separator . "Expires=" . $expires . "&Signature=" . $signature .
 "&Key-Pair-Id=" . $key_pair_id;
    }
    // not using a canned policy, include the policy itself in the stream name
    else {
        $result .= $separator . "Policy=" . $policy . "&Signature=" . $signature .
 "&Key-Pair-Id=" . $key_pair_id;
    }

    // new lines would break us, so remove them
    return str_replace('\n', '', $result);
}


function get_canned_policy_stream_name($video_path, $private_key_filename,
 $key_pair_id, $expires) {
    // this policy is well known by CloudFront, but you still need to sign it, since it
 contains your parameters
```

```
    $canned_policy = '{"Statement":[{"Resource":"' . $video_path . '","Condition":
{"DateLessThan":{"AWS:EpochTime":'. $expires . '}}}]}';
    // the policy contains characters that cannot be part of a URL, so we base64 encode
 it
    $encoded_policy = url_safe_base64_encode($canned_policy);
    // sign the original policy, not the encoded version
    $signature = rsa_sha1_sign($canned_policy, $private_key_filename);
    // make the signature safe to be included in a URL
    $encoded_signature = url_safe_base64_encode($signature);

    // combine the above into a stream name
    $stream_name = create_stream_name($video_path, null, $encoded_signature,
 $key_pair_id, $expires);
    // URL-encode the query string characters
    return $stream_name;
}

function get_custom_policy_stream_name($video_path, $private_key_filename,
 $key_pair_id, $policy) {
    // the policy contains characters that cannot be part of a URL, so we base64 encode
 it
    $encoded_policy = url_safe_base64_encode($policy);
    // sign the original policy, not the encoded version
    $signature = rsa_sha1_sign($policy, $private_key_filename);
    // make the signature safe to be included in a URL
    $encoded_signature = url_safe_base64_encode($signature);

    // combine the above into a stream name
    $stream_name = create_stream_name($video_path, $encoded_policy, $encoded_signature,
 $key_pair_id, null);
    // URL-encode the query string characters
    return $stream_name;
}


// Path to your private key.  Be very careful that this file is not accessible
// from the web!

$private_key_filename = '/home/test/secure/example-priv-key.pem';
$key_pair_id = 'K2JCJMDEHXQW5F';

// Make sure you have "Restrict viewer access" enabled on this path behaviour and using
 the above Trusted key groups (recommended).
$video_path = 'https://example.com/secure/example.mp4';
```

```
$expires = time() + 300; // 5 min from now
$canned_policy_stream_name = get_canned_policy_stream_name($video_path,
 $private_key_filename, $key_pair_id, $expires);

// Get the viewer real IP from the x-forwarded-for header as $_SERVER['REMOTE_ADDR']
 will return viewer facing IP. An alternative option is to use CloudFront-Viewer-
Address header. Note that this header is a trusted CloudFront immutable header. Example
 format: IP:PORT ("CloudFront-Viewer-Address": "1.2.3.4:12345")
$client_ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
$policy =
'{'.
    '"Statement":['.
        '{'.
            '"Resource":"'. $video_path . '",'.
            '"Condition":{'.
                '"IpAddress":{"AWS:SourceIp":"' . $client_ip . '/32"},'.
                '"DateLessThan":{"AWS:EpochTime":' . $expires . '}'.
            '}'.
        '}'.
    ']' .
    '}';
$custom_policy_stream_name = get_custom_policy_stream_name($video_path,
 $private_key_filename, $key_pair_id, $policy);

?>

<html>

<head>
    <title>CloudFront</title>
</head>

<body>
    <h1>Amazon CloudFront</h1>
    <h2>Canned Policy</h2>
    <h3>Expires at <?php echo gmdate('Y-m-d H:i:s T', $expires); ?></h3>
    <br />

    <div id='canned'>The canned policy video will be here: <br>

        <video width="640" height="360" autoplay muted controls>
        <source src="<?php echo $canned_policy_stream_name; ?>" type="video/mp4">
        Your browser does not support the video tag.
```

```
        </video>
    </div>

    <h2>Custom Policy</h2>
    <h3>Expires at <?php echo gmdate('Y-m-d H:i:s T', $expires); ?> only viewable by IP
 <?php echo $client_ip; ?></h3>
    <div id='custom'>The custom policy video will be here: <br>

         <video width="640" height="360" autoplay muted controls>
         <source src="<?php echo $custom_policy_stream_name; ?>" type="video/mp4">
         Your browser does not support the video tag.
        </video>
    </div>

</body>

</html>
```

For additional URL signature examples, see the following topics:

- Create a URL signature using Perl
- Create a URL signature using C# and the .NET Framework
- Create a URL signature using Java

Instead of using signed URLs to create the signature, you can use signed cookies. For more information, see Create signed cookies using PHP.

Create a URL signature using C# and the .NET Framework

The C# examples in this section implement an example application that demonstrates how to create the signatures for CloudFront private distributions using canned and custom policy statements. The examples include utility functions based on the AWS SDK for .NET that can be useful in .NET applications.

You can also create signed URLs and signed cookies by using the SDK for .NET. In the *SDK for .NET API Reference*, see the following topics:

- **Signed URLs** – AmazonCloudFrontUrlSigner
- **Signed cookies** – AmazonCloudFrontCookieSigner

To download the code, go to Signature Code in C#.

Notes

- The `AmazonCloudFrontUrlSigner` and `AmazonCloudFrontCookieSigner` classes have moved to a separate package. For more information about using them, see CookieSigner and UrlSigner in the *AWS SDK for .NET (V4) Developer Guide*.
- Creating a URL signature is just one part of the process of serving private content using a signed URL. For more information, see Use signed URLs. For more information about using signed cookies, see Use signed cookies.
- In the RSA signing call, note that SHA1 can be replaced with SHA256 in the hash algorithm parameter.

Use an RSA key in the .NET Framework

To use an RSA key in the .NET Framework, you must convert the AWS supplied .pem file to the XML format that the .NET Framework uses.

After conversion, the RSA private key file is in the following format:

Example: RSA private key in the XML .NET Framework format

```
<RSAKeyValue>
  <Modulus>
    wO5IvYCP5UcoCKDo1dcspoMehWBZcyfs9QEzGi6Oe5y+ewGr1oW+vB2GPB
    ANBiVPcUHTFWhwaIBd3oglmF0lGQljP/jOfmXHUK2kUUnLnJp+oOBL2NiuFtqcW6h/L5lIpD8Yq+NRHg
    Ty4zDsyr2880MvXv88yEFURCkqEXAMPLE=
  </Modulus>
  <Exponent>AQAB</Exponent>
  <P>
    5bmKDaTz
    npENGVqz4Cea8XPH+sxt+2VaAwYnsarVUoSBeVt8WLloVuZGG9IZYmH5KteXEu7fZveYd9UEXAMPLE==
  </P>
  <Q>
    1v9l/WN1a1N3rOK4VGoCokx7kR2SyTMSbZgF9IWJNOugR/WZw7HTnjipO3c9dy1Ms9pUKwUF4
    6d7049EXAMPLE==
  </Q>
  <DP>
    RgrSKuLWXMyBH+/l1Dx/I4tXuAJIrlPyo+VmiOc7b5NzHptkSHEPfR9s1
    OK0VqjknclqCJ3Ig86OMEtEXAMPLE==
```

```
  </DP>
  <DQ>
    pjPjvSFw+RoaTu0pgCA/jwW/FGyfN6iim1RFbkT4
    z49DZb2IM885f3vf35eLTaEYRYUHQgZtChNEV0TEXAMPLE==
  </DQ>
  <InverseQ>
    nkvOJTg5QtGNgWb9i
    cVtzrL/1pFEOHbJXwEJdU99N+7sMK+1066DL/HSBUCD63qD4USpnf0myc24in0EXAMPLE==</InverseQ>
  <D>
      Bc7mp7XYHynuPZxChjWNJZIq+A73gm0ASDv6At7F8Vi9r0xUlQe/v0AQS3ycN8QlyR4XMbzMLYk
      3yjxFDXo4ZKQtOGzLGteCU2srANiLv26/imXA8FVidZftTAtLviWQZBVPTeYIA69ATUYPEq0a5u5wjGy
      UOij9OWyuEXAMPLE=
   </D>
</RSAKeyValue>
```

Canned policy signing method in C#

The following C# code creates a signed URL that uses a canned policy by doing the following:

- Creates a policy statement.
- Hashes the policy statement using SHA1, and signs the result using RSA and the private key whose corresponding public key is in a trusted key group.
- Base64-encodes the hashed and signed policy statement and replaces special characters to make the string safe to use as a URL request parameter.
- Concatenates the values.

For the complete implementation, see the example at Signature Code in C#.

> **Note**
>
> The `keyId` is returned when you upload a public key to CloudFront. For more information, see
>
> 6
>
> &Key-Pair-Id.

Example: Canned policy signing method in C#

```
public static string ToUrlSafeBase64String(byte[] bytes)
{
```

```
    return System.Convert.ToBase64String(bytes)
        .Replace('+', '-')
        .Replace('=', '_')
        .Replace('/', '~');
}

public static string CreateCannedPrivateURL(string urlString,
    string durationUnits, string durationNumber, string pathToPolicyStmnt,
    string pathToPrivateKey, string keyId)
{
    // args[] 0-thisMethod, 1-resourceUrl, 2-seconds-minutes-hours-days
    // to expiration, 3-numberOfPreviousUnits, 4-pathToPolicyStmnt,
    // 5-pathToPrivateKey, 6-keyId

    TimeSpan timeSpanInterval = GetDuration(durationUnits, durationNumber);

    // Create the policy statement.
    string strPolicy = CreatePolicyStatement(pathToPolicyStmnt,
        urlString,
        DateTime.Now,
        DateTime.Now.Add(timeSpanInterval),
        "0.0.0.0/0");
    if ("Error!" == strPolicy) return "Invalid time frame." +
        "Start time cannot be greater than end time.";

    // Copy the expiration time defined by policy statement.
    string strExpiration = CopyExpirationTimeFromPolicy(strPolicy);

    // Read the policy into a byte buffer.
    byte[] bufferPolicy = Encoding.ASCII.GetBytes(strPolicy);

    // Initialize the SHA1CryptoServiceProvider object and hash the policy data.
    using (SHA1CryptoServiceProvider
        cryptoSHA1 = new SHA1CryptoServiceProvider())
    {
        bufferPolicy = cryptoSHA1.ComputeHash(bufferPolicy);

        // Initialize the RSACryptoServiceProvider object.
        RSACryptoServiceProvider providerRSA = new RSACryptoServiceProvider();
        XmlDocument xmlPrivateKey = new XmlDocument();

        // Load your private key, which you created by converting your
        // .pem file to the XML format that the .NET framework uses.
        // Several tools are available.
```

```
        xmlPrivateKey.Load(pathToPrivateKey);

        // Format the RSACryptoServiceProvider providerRSA and
        // create the signature.
        providerRSA.FromXmlString(xmlPrivateKey.InnerXml);
        RSAPKCS1SignatureFormatter rsaFormatter =
            new RSAPKCS1SignatureFormatter(providerRSA);
        rsaFormatter.SetHashAlgorithm("SHA1");
        byte[] signedPolicyHash = rsaFormatter.CreateSignature(bufferPolicy);

        // Convert the signed policy to URL-safe base64 encoding and
        // replace unsafe characters + = / with the safe characters - _ ~
        string strSignedPolicy = ToUrlSafeBase64String(signedPolicyHash);

        // Concatenate the URL, the timestamp, the signature,
        // and the key pair ID to form the signed URL.
        return urlString +
            "?Expires=" +
            strExpiration +
            "&Signature=" +
            strSignedPolicy +
            "&Key-Pair-Id=" +
            keyId;
    }
}
```

Custom policy signing method in C#

The following C# code creates a signed URL that uses a custom policy by doing the following:

1. Creates a policy statement.
2. Base64-encodes the policy statement and replaces special characters to make the string safe to use as a URL request parameter.
3. Hashes the policy statement using SHA1, and encrypts the result using RSA and the private key whose corresponding public key is in a trusted key group.
4. Base64-encodes the hashed policy statement and replaces special characters to make the string safe to use as a URL request parameter.
5. Concatenates the values.

For the complete implementation, see the example at Signature Code in C#.

Note

The `keyId` is returned when you upload a public key to CloudFront. For more information, see

6

&Key-Pair-Id.

Example: Custom policy signing method in C#

```
public static string ToUrlSafeBase64String(byte[] bytes)
{
    return System.Convert.ToBase64String(bytes)
        .Replace('+', '-')
        .Replace('=', '_')
        .Replace('/', '~');
}

public static string CreateCustomPrivateURL(string urlString,
    string durationUnits, string durationNumber, string startIntervalFromNow,
    string ipaddress, string pathToPolicyStmnt, string pathToPrivateKey,
    string keyId)
{
    // args[] 0-thisMethod, 1-resourceUrl, 2-seconds-minutes-hours-days
    // to expiration, 3-numberOfPreviousUnits, 4-starttimeFromNow,
    // 5-ip_address, 6-pathToPolicyStmt, 7-pathToPrivateKey, 8-keyId

    TimeSpan timeSpanInterval = GetDuration(durationUnits, durationNumber);
    TimeSpan timeSpanToStart = GetDurationByUnits(durationUnits,
        startIntervalFromNow);
    if (null == timeSpanToStart)
        return "Invalid duration units." +
            "Valid options: seconds, minutes, hours, or days";

    string strPolicy = CreatePolicyStatement(
        pathToPolicyStmnt, urlString, DateTime.Now.Add(timeSpanToStart),
        DateTime.Now.Add(timeSpanInterval), ipaddress);

    // Read the policy into a byte buffer.
    byte[] bufferPolicy = Encoding.ASCII.GetBytes(strPolicy);

    // Convert the policy statement to URL-safe base64 encoding and
```

```
    // replace unsafe characters + = / with the safe characters - _ ~

    string urlSafePolicy = ToUrlSafeBase64String(bufferPolicy);

    // Initialize the SHA1CryptoServiceProvider object and hash the policy data.
    byte[] bufferPolicyHash;
    using (SHA1CryptoServiceProvider cryptoSHA1 =
        new SHA1CryptoServiceProvider())
    {
        bufferPolicyHash = cryptoSHA1.ComputeHash(bufferPolicy);

        // Initialize the RSACryptoServiceProvider object.
        RSACryptoServiceProvider providerRSA = new RSACryptoServiceProvider();
        XmlDocument xmlPrivateKey = new XmlDocument();

        // Load your private key, which you created by converting your
        // .pem file to the XML format that the .NET framework uses.
        // Several tools are available.
        xmlPrivateKey.Load(pathToPrivateKey);

        // Format the RSACryptoServiceProvider providerRSA
        // and create the signature.
        providerRSA.FromXmlString(xmlPrivateKey.InnerXml);
        RSAPKCS1SignatureFormatter RSAFormatter =
            new RSAPKCS1SignatureFormatter(providerRSA);
        RSAFormatter.SetHashAlgorithm("SHA1");
        byte[] signedHash = RSAFormatter.CreateSignature(bufferPolicyHash);

        // Convert the signed policy to URL-safe base64 encoding and
        // replace unsafe characters + = / with the safe characters - _ ~
        string strSignedPolicy = ToUrlSafeBase64String(signedHash);

        return urlString +
            "?Policy=" +
            urlSafePolicy +
            "&Signature=" +
            strSignedPolicy +
            "&Key-Pair-Id=" +
            keyId;
    }
}
```

Utility methods for signature generation

The following methods get the policy statement from a file and parse time intervals for signature generation.

Example: Utility methods for signature generation

```
public static string CreatePolicyStatement(string policyStmnt,
   string resourceUrl,
   DateTime startTime,
   DateTime endTime,
   string ipAddress)

{
   // Create the policy statement.
   FileStream streamPolicy = new FileStream(policyStmnt, FileMode.Open,
 FileAccess.Read);
   using (StreamReader reader = new StreamReader(streamPolicy))
   {
      string strPolicy = reader.ReadToEnd();

      TimeSpan startTimeSpanFromNow = (startTime - DateTime.Now);
      TimeSpan endTimeSpanFromNow = (endTime - DateTime.Now);
      TimeSpan intervalStart =
         (DateTime.UtcNow.Add(startTimeSpanFromNow)) -
         new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
      TimeSpan intervalEnd =
         (DateTime.UtcNow.Add(endTimeSpanFromNow)) -
         new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);

      int startTimestamp = (int)intervalStart.TotalSeconds; // START_TIME
      int endTimestamp = (int)intervalEnd.TotalSeconds;  // END_TIME

      if (startTimestamp > endTimestamp)
         return "Error!";

      // Replace variables in the policy statement.
      strPolicy = strPolicy.Replace("RESOURCE", resourceUrl);
      strPolicy = strPolicy.Replace("START_TIME", startTimestamp.ToString());
      strPolicy = strPolicy.Replace("END_TIME", endTimestamp.ToString());
      strPolicy = strPolicy.Replace("IP_ADDRESS", ipAddress);
      strPolicy = strPolicy.Replace("EXPIRES", endTimestamp.ToString());
      return strPolicy;
   }
```

```
}

public static TimeSpan GetDuration(string units, string numUnits)
{
   TimeSpan timeSpanInterval = new TimeSpan();
   switch (units)
   {
      case "seconds":
         timeSpanInterval = new TimeSpan(0, 0, 0, int.Parse(numUnits));
         break;
      case "minutes":
         timeSpanInterval = new TimeSpan(0, 0, int.Parse(numUnits), 0);
         break;
      case "hours":
         timeSpanInterval = new TimeSpan(0, int.Parse(numUnits), 0 ,0);
         break;
      case "days":
         timeSpanInterval = new TimeSpan(int.Parse(numUnits),0 ,0 ,0);
         break;
      default:
         Console.WriteLine("Invalid time units;" +
            "use seconds, minutes, hours, or days");
         break;
   }
   return timeSpanInterval;
}

private static TimeSpan GetDurationByUnits(string durationUnits,
   string startIntervalFromNow)
{
   switch (durationUnits)
   {
      case "seconds":
         return new TimeSpan(0, 0, int.Parse(startIntervalFromNow));
      case "minutes":
         return new TimeSpan(0, int.Parse(startIntervalFromNow), 0);
      case "hours":
         return new TimeSpan(int.Parse(startIntervalFromNow), 0, 0);
      case "days":
         return new TimeSpan(int.Parse(startIntervalFromNow), 0, 0, 0);
      default:
         return new TimeSpan(0, 0, 0, 0);
   }
}
```

```
public static string CopyExpirationTimeFromPolicy(string policyStatement)
{
   int startExpiration = policyStatement.IndexOf("EpochTime");
   string strExpirationRough = policyStatement.Substring(startExpiration +
      "EpochTime".Length);
   char[] digits = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

   List<char> listDigits = new List<char>(digits);
   StringBuilder buildExpiration = new StringBuilder(20);

   foreach (char c in strExpirationRough)
   {
      if (listDigits.Contains(c))
         buildExpiration.Append(c);
   }
   return buildExpiration.ToString();
}
```

See also

- Create a URL signature using Perl
- Create a URL signature using PHP
- Create a URL signature using Java

Create a URL signature using Java

In addition to the following code example, you can use the `CloudFrontUrlSigner` utility class in the AWS SDK for Java (version 1) to create CloudFront signed URLs.

For more examples, see Create signed URLs and cookies using an AWS SDK in the *AWS SDK Code Examples Code Library*.

Notes

- Creating a signed URL is just one part of the process of serving private content with CloudFront. For more information about the entire process, see Use signed URLs.
- In the `Signature.getInstance` call, note that `SHA1withRSA` can be replaced with `SHA256withRSA`.

Example Java policy and signature encryption methods

```
package org.example;

import java.time.Instant;
import java.time.temporal.ChronoUnit;
import software.amazon.awssdk.services.cloudfront.CloudFrontUtilities;
import software.amazon.awssdk.services.cloudfront.model.CannedSignerRequest;
import software.amazon.awssdk.services.cloudfront.url.SignedUrl;

public class Main {

    public static void main(String[] args) throws Exception {
        CloudFrontUtilities cloudFrontUtilities = CloudFrontUtilities.create();
        Instant expirationDate = Instant.now().plus(7, ChronoUnit.DAYS);
        String resourceUrl = "https://a1b2c3d4e5f6g7.cloudfront.net";
        String keyPairId = "K1UA3WV15I7JSD";
        CannedSignerRequest cannedRequest = CannedSignerRequest.builder()
                .resourceUrl(resourceUrl)
                .privateKey(new java.io.File("/path/to/private_key.pem").toPath())
                .keyPairId(keyPairId)
                .expirationDate(expirationDate)
                .build();
        SignedUrl signedUrl =
 cloudFrontUtilities.getSignedUrlWithCannedPolicy(cannedRequest);
        String url = signedUrl.url();
        System.out.println(url);

    }
}
```

Example Canned Policy Signing with SHA256 in Java

```
package org.example;

import java.io.File;
import java.nio.file.Files;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;
import java.time.Instant;
import java.time.temporal.ChronoUnit;
```

```
import java.util.Base64;

public class Main {

    public static void main(String[] args) throws Exception {
        String resourceUrl = "https://a1b2c3d4e5f6g7.cloudfront.net/myfile.html";
        String keyPairId = "K1UA3WV15I7JSD";
        Instant expiration = Instant.now().plus(7, ChronoUnit.DAYS);
        PrivateKey privateKey = loadPrivateKey("/path/to/private_key.der");

        System.out.println(createSignedUrl(resourceUrl, keyPairId, privateKey,
 expiration, "SHA1"));
        System.out.println(createSignedUrl(resourceUrl, keyPairId, privateKey,
 expiration, "SHA256"));
    }

    static String createSignedUrl(String resourceUrl, String keyPairId,
                                  PrivateKey privateKey, Instant expiration,
                                  String hashAlgorithm) throws Exception {
        long epochSeconds = expiration.getEpochSecond();

        String policy = "{\"Statement\":[{\"Resource\":\"" + resourceUrl
                + "\",\"Condition\":{\"DateLessThan\":{\"AWS:EpochTime\":" +
 epochSeconds + "}}}]}";

        String jcaAlgorithm = hashAlgorithm.equals("SHA256") ? "SHA256withRSA" :
 "SHA1withRSA";

        Signature sig = Signature.getInstance(jcaAlgorithm);
        sig.initSign(privateKey);
        sig.update(policy.getBytes("UTF-8"));
        String signature = base64UrlEncode(sig.sign());

        String url = resourceUrl
                + (resourceUrl.contains("?") ? "&" : "?")
                + "Expires=" + epochSeconds
                + "&Signature=" + signature
                + "&Key-Pair-Id=" + keyPairId;

        if (hashAlgorithm.equals("SHA256")) {
            url += "&Hash-Algorithm=SHA256";
        }

        return url;
```

```
    }

    static String base64UrlEncode(byte[] bytes) {
        return Base64.getEncoder().encodeToString(bytes)
                .replace('+', '-')
                .replace('=', '_')
                .replace('/', '~');
    }

    static PrivateKey loadPrivateKey(String path) throws Exception {
        byte[] keyBytes = Files.readAllBytes(new File(path).toPath());
        return KeyFactory.getInstance("RSA")
                .generatePrivate(new PKCS8EncodedKeySpec(keyBytes));
    }
}
```

See also:

- Create a URL signature using Perl
- Create a URL signature using PHP
- Create a URL signature using C# and the .NET Framework

Restrict access to an AWS origin

You can configure CloudFront and some AWS origins in a way that provides the following benefits:

- Restricts access to the AWS origin so that it's not publicly accessible.
- Makes sure that viewers (users) can access the content in the AWS origin only through the specified CloudFront distribution. This prevents viewers from accessing the content directly from the origin, or through an unintended CloudFront distribution.

To do this, configure CloudFront to send authenticated requests to your AWS origin, and configure the AWS origin to only allow access to authenticated requests from CloudFront. For more information, see the following topics for compatible types of AWS origins.

Topics

- Restrict access to an AWS Elemental MediaPackage v2 origin
- Restrict access to an AWS Elemental MediaStore origin

- Restrict access to an AWS Lambda function URL origin
- Restrict access to an AWS Lambda Web Function origin
- Restrict access to an Amazon S3 origin
- Restrict access with VPC origins
- Restrict access to an Amazon S3 Multi-Region Access Point origin

Restrict access to an AWS Elemental MediaPackage v2 origin

CloudFront provides *origin access control* (OAC) for restricting access to a MediaPackage v2 origin.

> **Note**
>
> CloudFront OAC only supports MediaPackage v2. MediaPackage v1 isn't supported.

Topics

- Creating a new OAC
- Advanced settings for origin access control

Creating a new OAC

Complete the steps described in the following topics to set up a new OAC in CloudFront.

Topics

- Prerequisites
- Grant CloudFront permission to access the MediaPackage v2 origin
- Creating the OAC

Prerequisites

Before you create and set up OAC, you must have a CloudFront distribution with a MediaPackage v2 origin. For more information, see Use a MediaStore container or a MediaPackage channel.

Grant CloudFront permission to access the MediaPackage v2 origin

Before you create an OAC or set it up in a CloudFront distribution, make sure that CloudFront has permission to access the MediaPackage v2 origin. Do this after you create a CloudFront distribution, but before you add the OAC to the MediaPackage v2 origin in the distribution configuration.

Use an IAM policy to allow the CloudFront service principal (`cloudfront.amazonaws.com`) to access the origin. The `Condition` element in the policy allows CloudFront to access the MediaPackage v2 origin *only* when the request is on behalf of the CloudFront distribution that contains the MediaPackage v2 origin. This is the distribution with the MediaPackage v2 origin that you want to add OAC to.

Example: IAM policy that allows read-only access for a CloudFront distribution with OAC enabled

The following policy allows the CloudFront distribution (*E1PDK09ESKHJWT*) access to the MediaPackage v2 origin. The origin is the ARN specified for the `Resource` element.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontServicePrincipal",
            "Effect": "Allow",
            "Principal": {"Service": "cloudfront.amazonaws.com"},
            "Action": "mediapackagev2:GetObject",
            "Resource": "arn:aws-eusc:mediapackagev2:eusc-de-
east-1:123456789012:channelGroup/channel-group-name/channel/channel-name/
originEndpoint/origin_endpoint_name",
            "Condition": {
                "StringEquals": {"AWS:SourceArn": "arn:aws-
eusc:cloudfront::123456789012:distribution/E1PDK09ESKHJWT"}
            }
        }
    ]
}
```

Notes

- If you enabled the MQAR feature and origin access control (OAC), add the `mediapackagev2:GetHeadObject` action to the IAM policy. MQAR requires this permission to send HEAD requests to the MediaPackage v2 origin. For more information about MQAR, see Media quality-aware resiliency.
- If you create a distribution that doesn't have permission to your MediaPackage v2 origin, you can choose **Copy policy** from the CloudFront console and then choose **Update endpoint permissions**. You can then attach the copied permission to the endpoint. For more information, see Endpoint policy fields in the *AWS Elemental MediaPackage User Guide*.

Creating the OAC

To create an OAC, you can use the AWS Management Console, CloudFormation, the AWS CLI, or the CloudFront API.

Console

To create an OAC

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Origin access**.
3. Choose **Create control setting**.
4. On the **Create new OAC** form, do the following:
   a. Enter a **Name** and (optionally) a **Description** for the OAC.
   b. For **Signing behavior**, we recommend that you leave the default setting (**Sign requests (recommended)**). For more information, see the section called “Advanced settings for origin access control”.
5. For **Origin type**, choose **MediaPackage V2**.
6. Choose **Create**.

> **Tip**
>
> After you create the OAC, make note of the **Name**. You need this in the following procedure.

To add an OAC to a MediaPackage v2 origin in a distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose a distribution with a MediaPackage V2 origin that you want to add the OAC to, then choose the **Origins** tab.
3. Select the MediaPackage v2 origin that you want to add the OAC to, then choose **Edit**.
4. Select **HTTPS only** for your origin's **Protocol**.
5. From the **Origin access control** dropdown, choose the OAC name that you want to use.
6. Choose **Save changes**.

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the MediaPackage v2 origin.

CloudFormation

To create an OAC with CloudFormation, use the `AWS::CloudFront::OriginAccessControl` resource type. The following example shows the CloudFormation template syntax, in YAML format, for creating an OAC.

```
Type: AWS::CloudFront::OriginAccessControl
Properties:
  OriginAccessControlConfig:
      Description: An optional description for the origin access control
      Name: ExampleOAC
      OriginAccessControlOriginType: mediapackagev2
      SigningBehavior: always
      SigningProtocol: sigv4
```

For more information, see AWS::CloudFront::OriginAccessControl in the *AWS CloudFormation User Guide*.

CLI

To create an origin access control with the AWS Command Line Interface (AWS CLI), use the **aws cloudfront create-origin-access-control** command. You can use an input file to provide the input parameters for the command, rather than specifying each individual parameter as command line input.

To create an origin access control (CLI with input file)

1. Use the following command to create a file that's named `origin-access-control.yaml`. This file contains all of the input parameters for the **create-origin-access-control** command.

   ```
   aws cloudfront create-origin-access-control --generate-cli-skeleton yaml-input >
    origin-access-control.yaml
   ```

2. Open the `origin-access-control.yaml` file that you just created. Edit the file to add a name for the OAC, a description (optional), and change the `SigningBehavior` to `always`. Then save the file.

   For information about other OAC settings, see the section called “Advanced settings for origin access control”.

3. Use the following command to create the origin access control using the input parameters from the `origin-access-control.yaml` file.

   ```
   aws cloudfront create-origin-access-control --cli-input-yaml file://origin-
   access-control.yaml
   ```

   Make note of the `Id` value in the command output. You need it to add the OAC to a MediaPackage v2 origin in a CloudFront distribution.

To attach an OAC to a MediaPackage v2 origin in an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to add the OAC to. The distribution must have a MediaPackage v2 origin.

```
aws cloudfront get-distribution-config --id <CloudFront distribution ID> --
output yaml > dist-config.yaml
```

2. Open the file that's named `dist-config.yaml` that you just created. Edit the file, making the following changes:

   - In the `Origins` object, add the OAC's ID to the field that's named `OriginAccessControlId`.
   - Remove the value from the field that's named `OriginAccessIdentity`, if one exists.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the distribution to use the origin access control.

```
aws cloudfront update-distribution --id <CloudFront distribution ID> --cli-
input-yaml file://dist-config.yaml
```

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the MediaPackage v2 origin.

API

To create an OAC with the CloudFront API, use CreateOriginAccessControl. For more information about the fields that you specify in this API call, see the API reference documentation for your AWS SDK or other API client.

After you create an OAC you can attach it to a MediaPackage v2 origin in a distribution, using one of the following API calls:

- To attach it to an existing distribution, use UpdateDistribution.
- To attach it to a new distribution, use CreateDistribution.

For both of these API calls, provide the OAC ID in the `OriginAccessControlId` field, inside an origin. For more information about the other fields that you specify in these API calls, see All

distribution settings reference and the API reference documentation for your AWS SDK or other API client.

Advanced settings for origin access control

The CloudFront OAC feature includes advanced settings that are intended only for specific use cases. Use the recommended settings unless you have a specific need for the advanced settings.

OAC contains a setting named **Signing behavior** (in the console), or `SigningBehavior` (in the API, CLI, and CloudFormation). This setting provides the following options:

Always sign origin requests (recommended setting)

We recommend using this setting, named **Sign requests (recommended)** in the console, or `always` in the API, CLI, and CloudFormation. With this setting, CloudFront always signs all requests that it sends to the MediaPackage v2 origin.

Never sign origin requests

This setting is named **Do not sign requests** in the console, or `never` in the API, CLI, and CloudFormation. Use this setting to turn off OAC for all origins in all distributions that use this OAC. This can save time and effort compared to removing an OAC from all origins and distributions that use it, one by one. With this setting, CloudFront doesn't sign any requests that it sends to the MediaPackage v2 origin.

> **Warning**
>
> To use this setting, the MediaPackage v2 origin must be publicly accessible. If you use this setting with a MediaPackage v2 origin that's not publicly accessible, CloudFront can't access the origin. The MediaPackage v2 origin returns errors to CloudFront and CloudFront passes those errors on to viewers. For more information, see the example MediaPackage v2 policy for Policies and Permissions in MediaPackage in the *AWS Elemental MediaPackage User Guide*.

Don't override the viewer (client) `Authorization` header

This setting is named **Do not override authorization header** in the console, or `no-override` in the API, CLI, and CloudFormation. Use this setting when you want CloudFront to sign origin

requests only when the corresponding viewer request does not include an `Authorization` header. With this setting, CloudFront passes on the `Authorization` header from the viewer request when one is present, but signs the origin request (adding its own `Authorization` header) when the viewer request doesn't include an `Authorization` header.

> **Warning**
>
> To pass along the `Authorization` header from the viewer request, you *must* add the `Authorization` header to a cache policy for all cache behaviors that use MediaPackage v2 origins associated with this origin access control.

Restrict access to an AWS Elemental MediaStore origin

CloudFront provides *origin access control* (OAC) for restricting access to an AWS Elemental MediaStore origin.

Topics

- Create a new origin access control
- Advanced settings for origin access control

Create a new origin access control

Complete the steps described in the following topics to set up a new origin access control in CloudFront.

Topics

- Prerequisites
- Grant CloudFront permission to access the MediaStore origin
- Create the origin access control

Prerequisites

Before you create and set up origin access control, you must have a CloudFront distribution with a MediaStore origin.

Grant CloudFront permission to access the MediaStore origin

Before you create an origin access control or set it up in a CloudFront distribution, make sure that CloudFront has permission to access the MediaStore origin. Do this after creating a CloudFront distribution, but before adding the OAC to the MediaStore origin in the distribution configuration.

Use a MediaStore container policy to allow the CloudFront service principal (`cloudfront.amazonaws.com`) to access the origin. Use a `Condition` element in the policy to allow CloudFront to access the MediaStore container only when the request is on behalf of the CloudFront distribution that contains the MediaStore origin. This is the distribution with the MediaStore origin that you want to add OAC to.

The following are examples of MediaStore container policies that allow a CloudFront distribution to access a MediaStore origin.

Example MediaStore container policy that allows read-only access for a CloudFront distribution with OAC enabled

JSON

```
{
        "Version":"2012-10-17",
        "Statement": [
            {
                "Sid": "AllowCloudFrontServicePrincipalReadOnly",
                "Effect": "Allow",
                "Principal": {
                  "Service": "cloudfront.amazonaws.com"
                },
                "Action": [
                  "mediastore:GetObject"
                ],
                "Resource": "arn:aws-eusc:mediastore:eusc-de-
east-1:111122223333:container/<container name>/*",
                "Condition": {
                    "StringEquals": {
                      "AWS:SourceArn": "arn:aws-
eusc:cloudfront::111122223333:distribution/CloudFront-distribution-ID"
                    },
                    "Bool": {
                      "aws:SecureTransport": "true"
```

```
                    }
                }
            }
        ]
}
```

Example MediaStore container policy that allows read and write access for a CloudFront distribution with OAC enabled

JSON

```
{
        "Version":"2012-10-17",
        "Statement": [
            {
                "Sid": "AllowCloudFrontServicePrincipalReadWrite",
                "Effect": "Allow",
                "Principal": {
                  "Service": "cloudfront.amazonaws.com"
                },
                "Action": [
                  "mediastore:GetObject",
                  "mediastore:PutObject"
                ],
                "Resource": "arn:aws-eusc:mediastore:eusc-de-
east-1:111122223333:container/container-name/*",
                "Condition": {
                    "StringEquals": {
                      "AWS:SourceArn": "arn:aws-
eusc:cloudfront::111122223333:distribution/CloudFront-distribution-ID"
                    },
                    "Bool": {
                      "aws:SecureTransport": "true"
                    }
                }
            }
        ]
}
```

> **Note**
>
> To allow write access, you must configure **Allowed HTTP methods** to include PUT in your CloudFront distribution's behavior settings.

Create the origin access control

To create an OAC, you can use the AWS Management Console, CloudFormation, the AWS CLI, or the CloudFront API.

Console

To create an origin access control

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Origin access**.
3. Choose **Create control setting**.
4. On the **Create control setting** form, do the following:
   a. In the **Details** pane, enter a **Name** and (optionally) a **Description** for the origin access control.
   b. In the **Settings** pane, we recommend that you leave the default setting (**Sign requests (recommended)**). For more information, see the section called “Advanced settings for origin access control”.
5. Choose MediaStore from the **Origin type** dropdown.
6. Choose **Create**.

   After the OAC is created, make note of the **Name**. You need this in the following procedure.

To add an origin access control to a MediaStore origin in a distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose a distribution with a MediaStore origin that you want to add the OAC to, then choose the **Origins** tab.

3. Select the MediaStore origin that you want to add the OAC to, then choose **Edit**.
4. Select **HTTPS only** for your origin's **Protocol**.
5. From the **Origin access control** dropdown menu, choose the OAC that you want to use.
6. Choose **Save changes**.

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the MediaStore bucket origin.

CloudFormation

To create an origin access control (OAC) with CloudFormation, use the `AWS::CloudFront::OriginAccessControl` resource type. The following example shows the CloudFormation template syntax, in YAML format, for creating an origin access control.

```
Type: AWS::CloudFront::OriginAccessControl
Properties:
  OriginAccessControlConfig:
      Description: An optional description for the origin access control
      Name: ExampleOAC
      OriginAccessControlOriginType: mediastore
      SigningBehavior: always
      SigningProtocol: sigv4
```

For more information, see AWS::CloudFront::OriginAccessControl in the *AWS CloudFormation User Guide*.

CLI

To create an origin access control with the AWS Command Line Interface (AWS CLI), use the **aws cloudfront create-origin-access-control** command. You can use an input file to provide the input parameters for the command, rather than specifying each individual parameter as command line input.

To create an origin access control (CLI with input file)

1. Use the following command to create a file that's named `origin-access-control.yaml`. This file contains all of the input parameters for the **create-origin-access-control** command.

```
aws cloudfront create-origin-access-control --generate-cli-skeleton yaml-input >
 origin-access-control.yaml
```

2. Open the `origin-access-control.yaml` file that you just created. Edit the file to add a name for the OAC, a description (optional), and change the `SigningBehavior` to `always`. Then save the file.

   For information about other OAC settings, see the section called "Advanced settings for origin access control".

3. Use the following command to create the origin access control using the input parameters from the `origin-access-control.yaml` file.

   ```
   aws cloudfront create-origin-access-control --cli-input-yaml file://origin-
   access-control.yaml
   ```

   Make note of the `Id` value in the command output. You need it to add the OAC to a MediaStore origin in a CloudFront distribution.

To attach an OAC to a MediaStore origin in an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to add the OAC to. The distribution must have a MediaStore origin.

   ```
   aws cloudfront get-distribution-config --id <CloudFront distribution ID> --
   output yaml > dist-config.yaml
   ```

2. Open the file that's named `dist-config.yaml` that you just created. Edit the file, making the following changes:

   - In the `Origins` object, add the OAC's ID to the field that's named `OriginAccessControlId`.
   - Remove the value from the field that's named `OriginAccessIdentity`, if one exists.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

Save the file when finished.

3. Use the following command to update the distribution to use the origin access control.

```
aws cloudfront update-distribution --id <CloudFront distribution ID> --cli-
input-yaml file://dist-config.yaml
```

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the MediaStore origin.

API

To create an origin access control with the CloudFront API, use CreateOriginAccessControl. For more information about the fields that you specify in this API call, see the API reference documentation for your AWS SDK or other API client.

After you create an origin access control you can attach it to a MediaStore origin in a distribution, using one of the following API calls:

- To attach it to an existing distribution, use UpdateDistribution.
- To attach it to a new distribution, use CreateDistribution.

For both of these API calls, provide the origin access control ID in the `OriginAccessControlId` field, inside an origin. For more information about the other fields that you specify in these API calls, see All distribution settings reference and the API reference documentation for your AWS SDK or other API client.

Advanced settings for origin access control

The CloudFront origin access control feature includes advanced settings that are intended only for specific use cases. Use the recommended settings unless you have a specific need for the advanced settings.

Origin access control contains a setting named **Signing behavior** (in the console), or `SigningBehavior` (in the API, CLI, and CloudFormation). This setting provides the following options:

Always sign origin requests (recommended setting)

We recommend using this setting, named **Sign requests (recommended)** in the console, or `always` in the API, CLI, and CloudFormation. With this setting, CloudFront always signs all requests that it sends to the MediaStore origin.

Never sign origin requests

This setting is named **Do not sign requests** in the console, or `never` in the API, CLI, and CloudFormation. Use this setting to turn off origin access control for all origins in all distributions that use this origin access control. This can save time and effort compared to removing an origin access control from all origins and distributions that use it, one by one. With this setting, CloudFront does not sign any requests that it sends to the MediaStore origin.

> **Warning**
>
> To use this setting, the MediaStore origin must be publicly accessible. If you use this setting with a MediaStore origin that's not publicly accessible, CloudFront cannot access the origin. The MediaStore origin returns errors to CloudFront and CloudFront passes those errors on to viewers. For more information, see the example MediaStore container policy for Public read access over HTTPS.

Don't override the viewer (client) `Authorization` header

This setting is named **Do not override authorization header** in the console, or `no-override` in the API, CLI, and CloudFormation. Use this setting when you want CloudFront to sign origin requests only when the corresponding viewer request does not include an `Authorization` header. With this setting, CloudFront passes on the `Authorization` header from the viewer request when one is present, but signs the origin request (adding its own `Authorization` header) when the viewer request doesn't include an `Authorization` header.

> **Warning**
>
> To pass along the `Authorization` header from the viewer request, you *must* add the `Authorization` header to a cache policy for all cache behaviors that use MediaStore origins associated with this origin access control.

Restrict access to an AWS Lambda function URL origin

CloudFront provides *origin access control* (OAC) for restricting access to a Lambda function URL origin.

> **Note**
>
> If you use a Lambda Web Function as your CloudFront origin, see Restrict access to an AWS Lambda Web Function origin. Lambda Web Function origins require a different OAC configuration.

Topics

- Create a new OAC
- Advanced settings for origin access control
- Example template code

Create a new OAC

Complete the steps described in the following topics to set up a new OAC in CloudFront.

> **Important**
>
> If you use PUT or POST methods with your Lambda function URL, your users must compute the SHA256 of the body and include the payload hash value of the request body in the `x-amz-content-sha256` header when sending the request to CloudFront. Lambda doesn't support unsigned payloads.

Topics

- Prerequisites
- Grant CloudFront permission to access the Lambda function URL
- Create the OAC

Prerequisites

Before you create and set up OAC, you must have a CloudFront distribution with a Lambda function URL as the origin. To use OAC, you must specify `AWS_IAM` as the value for the `AuthType` parameter. For more information, see Use a Lambda function URL.

Grant CloudFront permission to access the Lambda function URL

Before you create an OAC or set it up in a CloudFront distribution, make sure that CloudFront has permission to access the Lambda function URL. Do this after you create a CloudFront distribution, but before you add the OAC to the Lambda function URL in the distribution configuration.

> **Note**
>
> To update the IAM policy for the Lambda function URL, you must use the AWS Command Line Interface (AWS CLI). Editing the IAM policy in the Lambda console isn't supported at this time.

The following AWS CLI command grants the CloudFront service principal (`cloudfront.amazonaws.com`) access to your Lambda function URL. The `Condition` element in the policy allows CloudFront to access Lambda *only* when the request is on behalf of the CloudFront distribution that contains the Lambda function URL. This is the distribution with the Lambda function URL origin that you want to add OAC to.

Example: AWS CLI command to update a policy to allow read-only access for a CloudFront distribution with OAC enabled

The following AWS CLI commands allow the CloudFront distribution (*E1PDK09ESKHJWT*) access to your Lambda *FUNCTION_URL_NAME*.

```
aws lambda add-permission \
--statement-id "AllowCloudFrontServicePrincipal" \
--action "lambda:InvokeFunctionUrl" \
--principal "cloudfront.amazonaws.com" \
--source-arn "arn:aws:cloudfront::123456789012:distribution/E1PDK09ESKHJWT" \
--function-name FUNCTION_URL_NAME
```

```
aws lambda add-permission \
```

```
--statement-id "AllowCloudFrontServicePrincipalInvokeFunction" \
--action "lambda:InvokeFunction" \
--principal "cloudfront.amazonaws.com" \
--source-arn "arn:aws:cloudfront::123456789012:distribution/E1PDK09ESKHJWT" \
--function-name FUNCTION_URL_NAME
```

> **Note**
>
> If you create a distribution and it doesn't have permission to your Lambda function URL, you can choose **Copy CLI command** from the CloudFront console, and then enter this command from your command line terminal. For more information, see Granting function access to AWS services in the *AWS Lambda Developer Guide*.

Create the OAC

To create an OAC, you can use the AWS Management Console, CloudFormation, the AWS CLI, or the CloudFront API.

Console

To create an OAC

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Origin access**.
3. Choose **Create control setting**.
4. On the **Create new OAC** form, do the following:
   a. Enter a **Name** and (optionally) a **Description** for the OAC.
   b. For **Signing behavior**, we recommend that you leave the default setting (**Sign requests (recommended)**). For more information, see the section called “Advanced settings for origin access control”.
5. For **Origin type**, choose **Lambda**.
6. Choose **Create**.

> **Tip**
>
> After you create the OAC, make note of the **Name**. You need this in the following procedure.

To add an origin access control to a Lambda function URL in a distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose a distribution with a Lambda function URL that you want to add the OAC to, then choose the **Origins** tab.
3. Select the Lambda function URL that you want to add the OAC to, and then choose **Edit**.
4. Select **HTTPS only** for your origin's **Protocol**.
5. From the **Origin access control** dropdown, choose the OAC name that you want to use.
6. Choose **Save changes**.

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the Lambda function URL.

CloudFormation

To create an OAC with CloudFormation, use the `AWS::CloudFront::OriginAccessControl` resource type. The following example shows the CloudFormation template syntax, in YAML format, for creating an OAC.

```
Type: AWS::CloudFront::OriginAccessControl
Properties:
  OriginAccessControlConfig:
      Description: An optional description for the origin access control
      Name: ExampleOAC
      OriginAccessControlOriginType: lambda
      SigningBehavior: always
      SigningProtocol: sigv4
```

For more information, see AWS::CloudFront::OriginAccessControl in the *AWS CloudFormation User Guide*.

CLI

To create an origin access control with the AWS Command Line Interface (AWS CLI), use the **aws cloudfront create-origin-access-control** command. You can use an input file to provide the input parameters for the command, rather than specifying each individual parameter as command line input.

To create an origin access control (CLI with input file)

1. Use the following command to create a file that's named `origin-access-control.yaml`. This file contains all of the input parameters for the **create-origin-access-control** command.

   ```
   aws cloudfront create-origin-access-control --generate-cli-skeleton yaml-input >
    origin-access-control.yaml
   ```

2. Open the `origin-access-control.yaml` file that you just created. Edit the file to add a name for the OAC, a description (optional), and change the `SigningBehavior` to `always`. Then save the file.

   For information about other OAC settings, see the section called “Advanced settings for origin access control”.

3. Use the following command to create the origin access control using the input parameters from the `origin-access-control.yaml` file.

   ```
   aws cloudfront create-origin-access-control --cli-input-yaml file://origin-
   access-control.yaml
   ```

   Make note of the `Id` value in the command output. You need it to add the OAC to a Lambda function URL in a CloudFront distribution.

To attach an OAC to a Lambda function URL in an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to add the OAC to. The distribution must have a Lambda function URL as the origin.

```
aws cloudfront get-distribution-config --id <CloudFront distribution ID> --
output yaml > dist-config.yaml
```

2. Open the file that's named `dist-config.yaml` that you just created. Edit the file, making the following changes:

   - In the `Origins` object, add the OAC's ID to the field that's named `OriginAccessControlId`.
   - Remove the value from the field that's named `OriginAccessIdentity`, if one exists.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the distribution to use the origin access control.

```
aws cloudfront update-distribution --id <CloudFront distribution ID> --cli-
input-yaml file://dist-config.yaml
```

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the Lambda function URL.

API

To create an OAC with the CloudFront API, use CreateOriginAccessControl. For more information about the fields that you specify in this API call, see the API reference documentation for your AWS SDK or other API client.

After you create an OAC you can attach it to a Lambda function URL in a distribution, using one of the following API calls:

- To attach it to an existing distribution, use UpdateDistribution.
- To attach it to a new distribution, use CreateDistribution.

For both of these API calls, provide the OAC ID in the `OriginAccessControlId` field, inside an origin. For more information about the other fields that you specify in these API calls, see and the API reference documentation for your AWS SDK or other API client.

Advanced settings for origin access control

The CloudFront OAC feature includes advanced settings that are intended only for specific use cases. Use the recommended settings unless you have a specific need for the advanced settings.

OAC contains a setting named **Signing behavior** (in the console), or `SigningBehavior` (in the API, CLI, and CloudFormation). This setting provides the following options:

Always sign origin requests (recommended setting)

We recommend using this setting, named **Sign requests (recommended)** in the console, or `always` in the API, CLI, and CloudFormation. With this setting, CloudFront always signs all requests that it sends to the Lambda function URL.

Never sign origin requests

This setting is named **Do not sign requests** in the console, or `never` in the API, CLI, and CloudFormation. Use this setting to turn off OAC for all origins in all distributions that use this OAC. This can save time and effort compared to removing an OAC from all origins and distributions that use it, one by one. With this setting, CloudFront doesn't sign any requests that it sends to the Lambda function URL.

> **Warning**
>
> To use this setting, the Lambda function URL must be publicly accessible. If you use this setting with a Lambda function URL that's not publicly accessible, CloudFront can't access the origin. The Lambda function URL returns errors to CloudFront and CloudFront passes those errors on to viewers. For more information, see Security and auth model for Lambda function URLs in the *AWS Lambda User Guide*.

Don't override the viewer (client) `Authorization` header

This setting is named **Do not override authorization header** in the console, or `no-override` in the API, CLI, and CloudFormation. Use this setting when you want CloudFront to sign origin requests only when the corresponding viewer request does not include an `Authorization` header. With this setting, CloudFront passes on the `Authorization` header from the viewer request when one is present, but signs the origin request (adding its own `Authorization` header) when the viewer request doesn't include an `Authorization` header.

Warning

- If you use this setting, you must specify the Signature Version 4 signing for the Lambda function URL instead of your CloudFront distribution's name or CNAME. When CloudFront forwards the `Authorization` header from the viewer request to the Lambda function URL, Lambda will validate the signature against the host of the Lambda URL domain. If the signature isn't based on the Lambda URL domain, the host in the signature won't match the host used by the Lambda URL origin. This means the request will fail, resulting in a signature validation error.
- To pass along the `Authorization` header from the viewer request, you *must* add the `Authorization` header to a cache policy for all cache behaviors that use Lambda function URLs associated with this origin access control.

Example template code

If your CloudFront origin is a Lambda function URL that's associated with an OAC, you can use the following Python script to upload files to the Lambda function with the `POST` method.

This code assumes that you configured the OAC with the default signing behavior set to **Always sign origin requests** and that you didn't select the **Do not override authorization header** setting.

This configuration allows the OAC to manage SigV4 authorization correctly with Lambda by using the Lambda hostname. The payload is signed by using SigV4 from the IAM identity that's authorized for the Lambda function URL, which is designated as the `AWS_IAM` type.

The template demonstrates how to handle signed payload hash values in the x-amz-content-sha256 header for `POST` requests from the client side. Specifically, this template is designed to manage form data payloads. The template enables secure file uploads to a Lambda function URL through CloudFront, and uses AWS authentication mechanisms to make sure that only authorized requests can access the Lambda function.

The code includes the following functionality:

- Meets the requirement for including the payload hash in the x-amz-content-sha256 header
- Uses SigV4 authentication for secure AWS service access
- Supports file uploads by using multi-part form data

- Includes error handling for request exceptions

```
import boto3
from botocore.auth import SigV4Auth
from botocore.awsrequest import AWSRequest
import requests
import hashlib
import os


def calculate_body_hash(body):
    return hashlib.sha256(body).hexdigest()


def sign_request(request, credentials, region, service):
    sigv4 = SigV4Auth(credentials, service, region)
    sigv4.add_auth(request)


def upload_file_to_lambda(cloudfront_url, file_path, region):
    # AWS credentials
    session = boto3.Session()
    credentials = session.get_credentials()

    # Prepare the multipart form-data
    boundary = "------------------------boundary"

    # Read file content
    with open(file_path, 'rb') as file:
        file_content = file.read()

    # Get the filename from the path
    filename = os.path.basename(file_path)

    # Prepare the multipart body
    body = (
        f'--{boundary}\r\n'
        f'Content-Disposition: form-data; name="file"; filename="{filename}"\r\n'
        f'Content-Type: application/octet-stream\r\n\r\n'
    ).encode('utf-8')
    body += file_content
    body += f'\r\n--{boundary}--\r\n'.encode('utf-8')
```

```
    # Calculate SHA256 hash of the entire body
    body_hash = calculate_body_hash(body)

    # Prepare headers
    headers = {
        'Content-Type': f'multipart/form-data; boundary={boundary}',
        'x-amz-content-sha256': body_hash
    }

    # Create the request
    request = AWSRequest(
        method='POST',
        url=cloudfront_url,
        data=body,
        headers=headers
    )

    # Sign the request
    sign_request(request, credentials, region, 'lambda')

    # Get the signed headers
    signed_headers = dict(request.headers)

    # Print request headers before sending
    print("Request Headers:")
    for header, value in signed_headers.items():
        print(f"{header}: {value}")

    try:
        # Send POST request with signed headers
        response = requests.post(
            cloudfront_url,
            data=body,
            headers=signed_headers
        )

        # Print response status and content
        print(f"\nStatus code: {response.status_code}")
        print("Response:", response.text)

        # Print response headers
        print("\nResponse Headers:")
        for header, value in response.headers.items():
```

```
            print(f"{header}: {value}")

    except requests.exceptions.RequestException as e:
        print(f"An error occurred: {e}")


# Usage
cloudfront_url = "https://d111111abcdef8.cloudfront.net"
file_path = r"filepath"
region = "us-east-1"  # example: "us-west-2"

upload_file_to_lambda(cloudfront_url, file_path, region)
```

Restrict access to an AWS Lambda Web Function origin

You can use origin access control (OAC) to restrict access to an AWS Lambda Web Function origin. A Lambda Web Function provides its own regional or global HTTPS endpoint, which you use as the origin domain for your CloudFront distribution. For more information about Lambda Web Functions, see AWS Lambda Web Functions in the *AWS Lambda Developer Guide*.

> **Note**
>
> OAC with Lambda Web Functions is not supported for global endpoints that include opt-in Regions. If the endpoint routes a request to an opt-in Region, CloudFront cannot sign the request for that Region. The origin then returns an HTTP 403 (Forbidden) error. All Regions associated with the global endpoint must be Regions that are enabled by default.

For information about using OAC with a Lambda function URL origin, see Restrict access to an AWS Lambda function URL origin.

Prerequisites

Before you create and set up OAC, you must have a CloudFront distribution with a Lambda Web Function as the origin. The origin domain name is the hostname of the Lambda Web Function's endpoint. CloudFront identifies the endpoint type from the hostname:

- Regional endpoint (`HomeRegion` and `PerRegion` endpoint types) – The hostname ends in `.lambda-web.`*`AWS-Region`*`.on.aws`

- Global endpoint (`MultiRegion` endpoint type) – The hostname ends in `.lambda-web.global.on.aws`

Your configuration must also meet the following requirements:

- **Endpoint auth type** – The endpoint's auth type must be `IamAuth`. With `IamAuth`, the endpoint uses AWS Identity and Access Management (IAM) to authenticate and authorize requests. For requests from CloudFront, authorization is based on the Lambda Web Function's resource-based policy. CloudFront signs requests on the viewer's behalf, and the endpoint validates the signature. For more information, see CloudFront integration in the *AWS Lambda Developer Guide*.
- **Cache policy** – If your Lambda Web Function uses the viewer's `Authorization` header, you must add the header to the cache policy. This requirement applies to every cache behavior that uses the origin. For `GET` and `HEAD` requests, CloudFront removes the `Authorization` header before forwarding the request unless the header is in the cache policy. For more information, see HTTP request headers and CloudFront behavior (custom and Amazon S3 origins).
- **Origin request policy** – Use an origin request policy that excludes the viewer's `Host` header. Lambda Web Function endpoints expect the `Host` header to contain the origin domain name, not the domain name of the CloudFront distribution. If the `Host` header contains the distribution's domain name, the endpoint returns an HTTP 404 (Not Found) error. If you use a managed origin request policy, use `AllViewerExceptHostHeader`, not `AllViewer`. For more information, see AllViewerExceptHostHeader.

> **Note**
>
> Lambda@Edge origin request and origin response triggers are not supported on cache behaviors that use a Lambda Web Function origin with OAC. This applies to regional and global endpoints, and to origins that are accessed directly or through an origin group. CloudFront rejects a distribution configuration that contains both.

Grant CloudFront permission to access the Lambda Web Function

Attach a resource-based policy to the Lambda Web Function that allows the CloudFront service principal (`cloudfront.amazonaws.com`) to invoke the endpoint. The IAM action for invoking an endpoint is `lambda:InvokeWebFunctionEndpoint`. The policy is attached to the Lambda Web Function, but each statement can scope the `Resource` to a specific endpoint ARN. Use a

`Condition` element to allow CloudFront to access the endpoint only when the request is on behalf of the CloudFront distribution that contains the origin.

Example Resource-based policy for CloudFront OAC

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontOACAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "lambda:InvokeWebFunctionEndpoint",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/my-
function/endpoint/prod",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn":
 "arn:aws:cloudfront::111122223333:distribution/EDFDVBD6EXAMPLE"
                }
            }
        }
    ]
}
```

To attach the resource-based policy, use the **put-resource-policy** command with the *function* ARN (not an endpoint ARN):

```
aws lambda-web put-resource-policy \
  --resource-arn "arn:aws:lambda:us-east-1:111122223333:web-function/my-function" \
  --policy file://policy.json \
  --region us-east-1
```

You can review or remove the policy with the **get-resource-policy** and **delete-resource-policy** commands.

> **Note**
>
> For endpoints that span more than one Region, Lambda replicates policy changes to all Regions that serve the endpoint. A new or updated policy can take a short time to take

effect in every Region. For more information, see Replication and eventual consistency in the *AWS Lambda Developer Guide*.

Create the origin access control

To create an origin access control (OAC), you can use the AWS Management Console, CloudFormation, the AWS CLI, or the CloudFront API.

Console

To create an origin access control

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Origin access**.
3. Choose **Create control setting**.
4. On the **Create control setting** form, do the following:
   a. In the **Details** pane, enter a **Name** and (optionally) a **Description** for the origin access control.
   b. Choose **Lambda** from the **Origin type** dropdown. This origin type is used for both Lambda function URLs and Lambda Web Functions.
   c. If your origin is a global endpoint (the hostname ends in `.lambda-web.global.on.aws`), select the **Use SigV4a signing protocol** option. For a regional endpoint (the hostname ends in `.lambda-web.`*`AWS-Region`*`.on.aws`), leave this option cleared.
   d. For **Signing behavior**, choose **Always sign, preserve viewer authorization**. This option appears under **Sign requests (recommended)**, which is selected by default. Lambda Web Function origins support only this signing behavior. For more information, see Signing behavior.
5. Choose **Create**.

To add an origin access control to a Lambda Web Function origin in a distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. Choose a distribution with a Lambda Web Function origin that you want to add the OAC to, then choose the **Origins** tab.
3. Select the Lambda Web Function origin that you want to add the OAC to, then choose **Edit**.
4. For **Origin access**, choose **Origin access control settings (recommended)**.
5. From the **Origin access control** dropdown menu, choose the OAC that you want to use.
6. Choose **Save changes**.

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the Lambda Web Function origin.

CLI

Use the **create-origin-access-control** command. For a regional endpoint (the hostname ends in `.lambda-web.`*`AWS-Region`*`.on.aws`), use the `sigv4` signing protocol:

```
aws cloudfront create-origin-access-control \
    --origin-access-control-config '{
        "Name": "my-lambda-web-oac",
        "Description": "OAC for Lambda Web Function origin",
        "SigningProtocol": "sigv4",
        "SigningBehavior": "always-amz-auth",
        "OriginAccessControlOriginType": "lambda"
    }'
```

For a global endpoint (the hostname ends in `.lambda-web.global.on.aws`), use the `sigv4a` signing protocol:

```
aws cloudfront create-origin-access-control \
    --origin-access-control-config '{
        "Name": "my-lambda-web-global-oac",
        "Description": "OAC for global Lambda Web Function origin",
        "SigningProtocol": "sigv4a",
        "SigningBehavior": "always-amz-auth",
        "OriginAccessControlOriginType": "lambda"
    }'
```

CloudFormation

Specify the following values in the `OriginAccessControlConfig`:

- `SigningProtocol`: `sigv4` (regional endpoint) or `sigv4a` (global endpoint)
- `SigningBehavior`: `always-amz-auth`
- `OriginAccessControlOriginType`: `lambda`

Example CloudFormation template

```
Type: AWS::CloudFront::OriginAccessControl
Properties:
  OriginAccessControlConfig:
    Description: An optional description for the origin access control
    Name: my-lambda-web-oac
    OriginAccessControlOriginType: lambda
    SigningBehavior: always-amz-auth
    SigningProtocol: sigv4a
```

For more information, see AWS::CloudFront::OriginAccessControl in the *AWS CloudFormation User Guide*.

OAC takes effect for every request only after the distribution finishes deploying. For an endpoint that spans more than one Region, the resource-based policy that you attached earlier must also finish replicating to every Region that serves the endpoint. Until both complete, some requests can receive a transient HTTP 403 response from the origin. These errors resolve automatically after the distribution finishes deploying and the policy finishes replicating.

Signing behavior

Lambda Web Function origins support only the `always-amz-auth` signing behavior (**Always sign, preserve viewer authorization** in the console). CloudFront always signs all requests that it sends to the Lambda Web Function, but it does not replace the `Authorization` header from the viewer request. Instead, CloudFront adds its signature in a dedicated CloudFront header and passes the viewer's `Authorization` header to the origin unmodified. With this behavior, your Lambda Web Function can receive the viewer's original `Authorization` header (for example, to perform its own authentication or authorization). CloudFront still signs every request that it sends to the origin. To forward the viewer's `Authorization` header, you must add it to a cache policy for every cache behavior that uses the origin.

The other signing behaviors (`always`, `never`, and `no-override`) are not supported for Lambda Web Function origins. The `always-amz-auth` signing behavior is not supported for any other origin type, including Lambda function URL origins.

For `PUT`, `POST`, and `PATCH` requests, CloudFront signs the origin request with an unsigned payload (`UNSIGNED-PAYLOAD`). The request body is not covered by the CloudFront signature. The signature still covers the request method, path, query string, and the headers that identify your distribution. The endpoint can therefore verify that each request comes from CloudFront on behalf of your distribution before it invokes your Lambda Web Function. The request body is protected in transit by the HTTPS connection between CloudFront and the endpoint. You do not need to include an `x-amz-content-sha256` header in requests to CloudFront. If a viewer sends one, CloudFront does not use it when signing the origin request.

Restrict access to an Amazon S3 origin

CloudFront provides two ways to send authenticated requests to an Amazon S3 origin: *origin access control* (OAC) and *origin access identity* (OAI). OAC helps you secure your origins, such as Amazon S3.

We *recommend* that you use OAC instead because it supports the following features:

- All Amazon S3 buckets in all AWS Regions, including opt-in Regions launched after December 2022
- Amazon S3 server-side encryption with AWS KMS (SSE-KMS)
- Dynamic requests (`PUT` and `DELETE`) to Amazon S3

OAI doesn't support these features or it requires extra workarounds in those scenarios. If you're already using OAI and want to migrate, see the section called “Migrating from origin access identity (OAI) to origin access control (OAC)”.

Notes

- When you use CloudFront OAC with Amazon S3 bucket origins, you must set **Amazon S3 Object Ownership** to **Bucket owner enforced**, the default for new Amazon S3 buckets. If you require ACLs, use the **Bucket owner preferred** setting to maintain control over objects uploaded by using CloudFront.

- If your origin is an Amazon S3 bucket configured as a website endpoint, you must set it up with CloudFront as a custom origin. That means you can't use OAC (or OAI). OAC doesn't support origin redirect by using Lambda@Edge.
- If you use an Amazon S3 Multi-Region Access Point as your CloudFront origin, see the section called "Restrict access to an Amazon S3 Multi-Region Access Point origin". S3 Multi-Region Access Points require a different OAC configuration.

The following topics describe how to use OAC with an Amazon S3 origin.

Topics

- the section called "Create a new origin access control"
- the section called "Delete a distribution with an OAC attached to an S3 bucket"
- the section called "Migrating from origin access identity (OAI) to origin access control (OAC)"
- the section called "Advanced settings for origin access control"

Create a new origin access control

Complete the steps described in the following topics to set up a new origin access control in CloudFront.

Topics

- Prerequisites
- Grant CloudFront permission to access the S3 bucket
- Create the origin access control

Prerequisites

Before you create and set up origin access control (OAC), you must have a CloudFront distribution with an Amazon S3 bucket origin. This origin must be a regular S3 bucket, not a bucket configured as a website endpoint. For more information about setting up a CloudFront distribution with an S3 bucket origin, see the section called "Get started with a standard distribution".

Important

When you use OAC to secure your Amazon S3 origin, communication between CloudFront and Amazon S3 is *always* through HTTPS, but only when you choose to *always sign requests*. You must choose **Sign requests (recommended)** in the console or specify `always` in the CloudFront API, AWS CLI, or CloudFormation.
If you choose either the **Do not sign requests** or **Do not override authorization header** option instead, CloudFront uses the connection protocol that you specified in the following policies:

- Viewer protocol policy
- Origin protocol policy (custom origins only)

For example, if you choose **Do not override authorization header** and want to use HTTPS between CloudFront and your Amazon S3 origin, use **Redirect HTTP to HTTPS** or **HTTPS only** for the viewer protocol policy.

Grant CloudFront permission to access the S3 bucket

Before you create an origin access control (OAC) or set it up in a CloudFront distribution, make sure that CloudFront has permission to access the S3 bucket origin. Do this after creating a CloudFront distribution, but before adding the OAC to the S3 origin in the distribution configuration.

Use an S3 bucket policy to allow the CloudFront service principal (`cloudfront.amazonaws.com`) to access the bucket. Use a `Condition` element in the policy to allow CloudFront to access the bucket only when the request is on behalf of the CloudFront distribution that contains the S3 origin. This is the distribution with the S3 origin that you want to add OAC to.

For information about adding or modifying a bucket policy, see Adding a bucket policy using the Amazon S3 console in the *Amazon S3 User Guide*.

The following are examples of S3 bucket policies that allow a CloudFront distribution with OAC enabled access to an S3 origin.

Example S3 bucket policy that allows read-only access for a CloudFront distribution with OAC enabled

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudFrontServicePrincipalReadOnly",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws-
eusc:cloudfront::111122223333:distribution/<CloudFront distribution ID>"
        }
      }
    }
  ]
}
```

Example S3 bucket policy that allows read and write access for a CloudFront distribution with OAC enabled

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCloudFrontServicePrincipalReadWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
```

```
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws-
eusc:cloudfront::111122223333:distribution/CloudFront-distribution-ID>"
        }
      }
    }
  ]
}
```

SSE-KMS

If the objects in the S3 bucket origin are encrypted using server-side encryption with AWS Key Management Service (SSE-KMS), you must make sure that the CloudFront distribution has permission to use the AWS KMS key. To give the CloudFront distribution permission to use the KMS key, add a statement to the KMS key policy. For information about how to modify a key policy, see Changing a key policy in the *AWS Key Management Service Developer Guide*.

Example KMS key policy statement

The following example shows an AWS KMS policy statement that allows the CloudFront distribution with OAC to access a KMS key for SSE-KMS.

```
{
    "Sid": "AllowCloudFrontServicePrincipalSSE-KMS",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "cloudfront.amazonaws.com"
        ]
     },
    "Action": [
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*",
```

```
    "Condition": {
            "StringEquals": {
                "AWS:SourceArn":
 "arn:aws:cloudfront::111122223333:distribution/<CloudFront distribution ID>"
            }
        }
}
```

Create the origin access control

To create an origin access control (OAC), you can use the AWS Management Console, CloudFormation, the AWS CLI, or the CloudFront API.

Console

To create an origin access control

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Origin access**.
3. Choose **Create control setting**.
4. On the **Create control setting** form, do the following:
   a. In the **Details** pane, enter a **Name** and (optionally) a **Description** for the origin access control.
   b. In the **Settings** pane, we recommend that you leave the default setting (**Sign requests (recommended)**). For more information, see the section called “Advanced settings for origin access control”.
5. Choose S3 from the **Origin type** dropdown.
6. Choose **Create**.

   After the OAC is created, make note of the **Name**. You need this in the following procedure.

To add an origin access control to an S3 origin in a distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. Choose a distribution with an S3 origin that you want to add the OAC to, then choose the **Origins** tab.
3. Select the S3 origin that you want to add the OAC to, then choose **Edit**.
4. For **Origin access**, choose **Origin access control settings (recommended)**.
5. From the **Origin access control** dropdown menu, choose the OAC that you want to use.
6. Choose **Save changes**.

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the S3 bucket origin.

CloudFormation

To create an origin access control (OAC) with CloudFormation, use the `AWS::CloudFront::OriginAccessControl` resource type. The following example shows the CloudFormation template syntax, in YAML format, for creating an origin access control.

```
Type: AWS::CloudFront::OriginAccessControl
Properties:
  OriginAccessControlConfig:
      Description: An optional description for the origin access control
      Name: ExampleOAC
      OriginAccessControlOriginType: s3
      SigningBehavior: always
      SigningProtocol: sigv4
```

For more information, see AWS::CloudFront::OriginAccessControl in the *AWS CloudFormation User Guide*.

CLI

To create an origin access control with the AWS Command Line Interface (AWS CLI), use the **aws cloudfront create-origin-access-control** command. You can use an input file to provide the input parameters for the command, rather than specifying each individual parameter as command line input.

To create an origin access control (CLI with input file)

1. Use the following command to create a file that's named `origin-access-control.yaml`. This file contains all of the input parameters for the **create-origin-access-control** command.

```
aws cloudfront create-origin-access-control --generate-cli-skeleton yaml-input >
 origin-access-control.yaml
```

2. Open the `origin-access-control.yaml` file that you just created. Edit the file to add a name for the OAC, a description (optional), and change the `SigningBehavior` to `always`. Then save the file.

   For information about other OAC settings, see the section called "Advanced settings for origin access control".

3. Use the following command to create the origin access control using the input parameters from the `origin-access-control.yaml` file.

   ```
   aws cloudfront create-origin-access-control --cli-input-yaml file://origin-
   access-control.yaml
   ```

   Make note of the `Id` value in the command output. You need it to add the OAC to an S3 bucket origin in a CloudFront distribution.

To attach an OAC to an S3 bucket origin in an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to add the OAC to. The distribution must have an S3 bucket origin.

   ```
   aws cloudfront get-distribution-config --id <CloudFront distribution ID> --
   output yaml > dist-config.yaml
   ```

2. Open the file that's named `dist-config.yaml` that you just created. Edit the file, making the following changes:

   - In the `Origins` object, add the OAC's ID to the field that's named `OriginAccessControlId`.
   - Remove the value from the field that's named `OriginAccessIdentity`, if one exists.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

Save the file when finished.

3. Use the following command to update the distribution to use the origin access control.

```
aws cloudfront update-distribution --id <CloudFront distribution ID> --cli-
input-yaml file://dist-config.yaml
```

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the S3 bucket origin.

API

To create an origin access control with the CloudFront API, use CreateOriginAccessControl. For more information about the fields that you specify in this API call, see the API reference documentation for your AWS SDK or other API client.

After you create an origin access control you can attach it to an S3 bucket origin in a distribution, using one of the following API calls:

- To attach it to an existing distribution, use UpdateDistribution.
- To attach it to a new distribution, use CreateDistribution.

For both of these API calls, provide the origin access control ID in the `OriginAccessControlId` field, inside an origin. For more information about the other fields that you specify in these API calls, see All distribution settings reference and the API reference documentation for your AWS SDK or other API client.

Delete a distribution with an OAC attached to an S3 bucket

If you need to delete a distribution with an OAC attached to an S3 bucket, you should delete the distribution before you delete the S3 bucket origin. Alternatively, include the Region in the origin domain name. If this isn't possible, you can remove the OAC from the distribution by switching to public before deletion. For more information, see Delete a distribution.

Migrating from origin access identity (OAI) to origin access control (OAC)

To migrate from a legacy origin access identity (OAI) to an origin access control (OAC), first update the S3 bucket origin to allow both the OAI and the distribution with OAC enabled to access the bucket's content. This makes sure that CloudFront never loses access to the bucket during the transition. To allow both OAI and the distribution with OAC enabled to access an S3 bucket, update the bucket policy to include two statements, one for each kind of principal.

The following example S3 bucket policy allows both an OAI and a distribution with OAC enabled to access an S3 origin.

Example S3 bucket policy that allows read-only access for an OAI and a CloudFront distribution with OAC enabled

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontServicePrincipalReadOnly",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws-eusc:s3:::<S3 bucket name>/*",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceArn": "arn:aws-
eusc:cloudfront::111122223333:distribution/<CloudFront distribution ID>"
                }
            }
        },
        {
            "Sid": "AllowLegacyOAIReadOnly",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-eusc:iam::cloudfront:user/CloudFront Origin
 Access Identity <origin access identity ID>"
            },
            "Action": "s3:GetObject",
```

```
            "Resource": "arn:aws-eusc:s3:::<S3 bucket name>/*"
        }
    ]
}
```

After you update the S3 origin's bucket policy to allow access to both OAI and OAC, you can update the distribution configuration to use OAC instead of OAI. For more information, see the section called “Create a new origin access control”.

After the distribution is fully deployed, you can remove the statement in the bucket policy that allows access to the OAI. For more information, see the section called “Grant CloudFront permission to access the S3 bucket”.

Advanced settings for origin access control

The CloudFront origin access control feature includes advanced settings that are intended only for specific use cases. Use the recommended settings unless you have a specific need for the advanced settings.

Origin access control contains a setting named **Signing behavior** (in the console), or `SigningBehavior` (in the API, CLI, and CloudFormation). This setting provides the following options:

Always sign origin requests (recommended setting)

We recommend using this setting, named **Sign requests (recommended)** in the console, or `always` in the API, CLI, and CloudFormation. With this setting, CloudFront always signs all requests that it sends to the S3 bucket origin.

Never sign origin requests

This setting is named **Do not sign requests** in the console, or `never` in the API, CLI, and CloudFormation. Use this setting to turn off origin access control for all origins in all distributions that use this origin access control. This can save time and effort compared to removing an origin access control from all origins and distributions that use it, one by one. With this setting, CloudFront does not sign any requests that it sends to the S3 bucket origin.

> **Warning**
>
> To use this setting, the S3 bucket origin must be publicly accessible. If you use this setting with an S3 bucket origin that's not publicly accessible, CloudFront cannot access

the origin. The S3 bucket origin returns errors to CloudFront and CloudFront passes those errors on to viewers.

Don't override the viewer (client) `Authorization` header

This setting is named **Do not override authorization header** in the console, or `no-override` in the API, CLI, and CloudFormation. Use this setting when you want CloudFront to sign origin requests only when the corresponding viewer request does not include an `Authorization` header. With this setting, CloudFront passes on the `Authorization` header from the viewer request when one is present, but signs the origin request (adding its own `Authorization` header) when the viewer request doesn't include an `Authorization` header.

> **Warning**
>
> To pass along the `Authorization` header from the viewer request, you *must* add the `Authorization` header to a cache policy for all cache behaviors that use S3 bucket origins associated with this origin access control.

Use an origin access identity (legacy, not recommended)

Overview of origin access identity

CloudFront *origin access identity* (OAI) provides similar functionality as *origin access control* (OAC), but it doesn't work for all scenarios. Specifically, OAI doesn't support:

- Amazon S3 buckets in all AWS Regions, including opt-in Regions
- Amazon S3 server-side encryption with AWS KMS (SSE-KMS)
- Dynamic requests (`PUT`, `POST`, or `DELETE`) to Amazon S3
- New AWS Regions launched after January 2023

> **Tip**
>
> We recommend that you use OAC instead. To set up OAC, see Create a new origin access control. For information about how to migrate from OAI to OAC, see the section called "Migrating from origin access identity (OAI) to origin access control (OAC)".

Give an origin access identity permission to read files in the Amazon S3 bucket

When you create an OAI or add one to a distribution with the CloudFront console, you can automatically update the Amazon S3 bucket policy to give the OAI permission to access your bucket. Alternatively, you can choose to manually create or update the bucket policy. Whichever method you use, you should still review the permissions to make sure that:

- Your CloudFront OAI can access files in the bucket on behalf of viewers who are requesting them through CloudFront.
- Viewers can't use Amazon S3 URLs to access your files outside of CloudFront.

> **Important**
>
> If you configure CloudFront to accept and forward all of the HTTP methods that CloudFront supports, make sure you give your CloudFront OAI the desired permissions. For example, if you configure CloudFront to accept and forward requests that use the `DELETE` method, configure your bucket policy to handle `DELETE` requests appropriately so viewers can delete only files that you want them to.

Use Amazon S3 bucket policies

You can give a CloudFront OAI access to files in an Amazon S3 bucket by creating or updating the bucket policy in the following ways:

- Using the Amazon S3 bucket's **Permissions** tab in the Amazon S3 console.
- Using PutBucketPolicy in the Amazon S3 API.
- Using the CloudFront console. When you add an OAI to your origin settings in the CloudFront console, you can choose **Yes, update the bucket policy** to tell CloudFront to update the bucket policy on your behalf.

If you update the bucket policy manually, make sure that you:

- Specify the correct OAI as the `Principal` in the policy.
- Give the OAI the permissions it needs to access objects on behalf of viewers.

For more information, see the following sections.

Specify an OAI as the `Principal` in a bucket policy

To specify an OAI as the `Principal` in an Amazon S3 bucket policy, use the OAI's Amazon Resource Name (ARN), which includes the OAI's ID. For example:

```
"Principal": {
    "AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity <origin
 access identity ID>"
}
```

Find the OAI ID in the CloudFront console under **Security**, **Origin access**, **Identities (legacy)**. Alternatively, use ListCloudFrontOriginAccessIdentities in the CloudFront API.

Give permissions to an OAI

To give the OAI the permissions to access objects in your Amazon S3 bucket, use actions in the policy that relate to specific Amazon S3 API operations. For example, the `s3:GetObject` action allows the OAI to read objects in the bucket. For more information, see the examples in the following section, or see Amazon S3 actions in the *Amazon Simple Storage Service User Guide*.

Amazon S3 bucket policy examples

The following examples show Amazon S3 bucket policies that allow CloudFront OAI to access an S3 bucket.

Find the OAI ID in the CloudFront console under **Security**, **Origin access**, **Identities (legacy)**. Alternatively, use ListCloudFrontOriginAccessIdentities in the CloudFront API.

Example Amazon S3 bucket policy that gives the OAI read access

The following example allows the OAI to read objects in the specified bucket (`s3:GetObject`).

JSON

```
{
    "Version":"2012-10-17",
    "Id": "PolicyForCloudFrontPrivateContent",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": "arn:aws-eusc:iam::cloudfront:user/CloudFront Origin
 Access Identity <origin access identity ID>"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws-eusc:s3:::<S3 bucket name>/*"
        }
    ]
}
```

Example Amazon S3 bucket policy that gives the OAI read and write access

The following example allows the OAI to read and write objects in the specified bucket (`s3:GetObject` and `s3:PutObject`). This allows viewers to upload files to your Amazon S3 bucket through CloudFront.

JSON

```
{
    "Version":"2012-10-17",
    "Id": "PolicyForCloudFrontPrivateContent",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-eusc:iam::cloudfront:user/CloudFront Origin
 Access Identity <origin access identity ID>"
            },
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws-eusc:s3:::<S3 bucket name>/*"
        }
    ]
}
```

Use Amazon S3 object ACLs (not recommended)

> **Important**
>
> We recommend using Amazon S3 bucket policies to give an OAI access to an S3 bucket. You can use access control lists (ACLs) as described in this section, but we don't recommend it. Amazon S3 recommends setting S3 Object Ownership to **bucket owner enforced**, which means that ACLs are disabled for the bucket and the objects in it. When you apply this setting for Object Ownership, you must use bucket policies to give access to the OAI (see the previous section).
>
> This following section is only for legacy use cases that require ACLs.

You can give a CloudFront OAI access to files in an Amazon S3 bucket by creating or updating the file's ACL in the following ways:

- Using the Amazon S3 object's **Permissions** tab in the Amazon S3 console.
- Using PutObjectAcl in the Amazon S3 API.

When you grant access to an OAI using an ACL, you must specify the OAI using its Amazon S3 canonical user ID. In the CloudFront console, you can find this ID under **Security**, **Origin access**, **Identities (legacy)**. If you're using the CloudFront API, use the value of the `S3CanonicalUserId` element that was returned when you created the OAI, or call ListCloudFrontOriginAccessIdentities in the CloudFront API.

Use an origin access identity in Amazon S3 regions that support only signature version 4 authentication

Newer Amazon S3 Regions require that you use Signature Version 4 for authenticated requests. (For the signature versions supported in each Amazon S3 Region, see Amazon Simple Storage Service endpoints and quotas in the *AWS General Reference*.) If you're using an origin access identity and if your bucket is in one of the Regions that requires Signature Version 4, note the following:

- `DELETE`, `GET`, `HEAD`, `OPTIONS`, and `PATCH` requests are supported without qualifications.
- `POST` requests are not supported.

Restrict access with VPC origins

You can use CloudFront to deliver content from applications that are hosted in your virtual private cloud (VPC) private subnets. You can use Application Load Balancers (ALBs), Network Load Balancers (NLBs), and EC2 instances in private subnets as VPC origins.

Here are some reasons why you might want to use VPC origins:

- **Security** – VPC origins is designed to enhance the security posture of your application by placing your load balancers and EC2 instances in private subnets, making CloudFront the single point of entry. User requests go from CloudFront to the VPC origins over a private, secure connection, providing additional security for your applications.
- **Management** – VPC origins reduces the operational overhead required for secure connectivity between CloudFront and origins. You can move your origins to private subnets with no public access, and you don't have to implement access control lists (ACLs) or other mechanisms to restrict access to your origins. This way, you don't have to invest in undifferentiated development work to secure your web applications with CloudFront.
- **Scalability and performance** – VPC origins helps you to secure your web applications, freeing up time to focus on growing your critical business applications while improving security and maintaining high performance and global scalability with CloudFront. VPC origins streamlines security management and reduces operational complexity so that you can use CloudFront as the single point of entry for your applications.

> **Tip**
>
> CloudFront supports sharing VPC origins across AWS accounts, whether they're in your organization or not. You can share VPC origins from the CloudFront console or use AWS Resource Access Manager (AWS RAM). For more information, see Working with shared resources in CloudFront.

Prerequisites

Before you create a VPC origin for your CloudFront distribution, you must complete the following:

VPC Configuration

Create a virtual private cloud (VPC) on Amazon VPC in one of the AWS Regions that are supported for VPC origins. For information about creating a VPC, see Create a VPC plus other VPC resources in the *Amazon VPC User Guide*. For a list of supported Regions, see Supported AWS Regions for VPC origins.

Your VPC must include the following:

- **Internet gateway** – You need to add an internet gateway to the VPC that has your VPC origin resources. The internet gateway is required to denote that the VPC can receive traffic from the internet. The internet gateway is not used for routing traffic to origins inside the subnet, and you don't need to update the routing policies.
- **Private subnet with at least one available IPv4 address (and, for dual-stack support, at least one available IPv6 address)** – CloudFront routes to your subnet by using a service-managed elastic network interface (ENI) that CloudFront creates after you define your VPC origin resource. You must have at least one available IPv4 address (and, for dual-stack support, at least one available IPv6 address) in your private subnet so that the ENI creation process can succeed. These addresses can be private, and there is no additional cost for them.

Origin Resources

In the private subnet, launch an Application Load Balancer, a Network Load Balancer, or an EC2 instance to use as your origin. The resource you launch must be fully deployed and in Active status before you can use it for a VPC origin.

Origin restrictions:

- Gateway Load Balancers cannot be added as origins
- Network Load Balancers with TLS listeners cannot be added as origins
- To be used as a VPC origin, a Network Load Balancer must have a security group attached to it

Security Group Configuration

Your VPC origin resources (Application Load Balancer, Network Load Balancer, or EC2 instance) must have a security group attached. When you create a VPC origin, CloudFront automatically creates a service-managed security group with the naming pattern `CloudFront-VPCOrigins-Service-SG`. This security group is fully managed by AWS, and should not be edited.

To allow traffic from CloudFront to reach your VPC origin, update the security group attached to your origin resource (ALB, NLB, or EC2 instance) to allow inbound traffic using one of the following methods:

- **Option 1:** Allow traffic from the CloudFront managed prefix list. For more information, see Use the CloudFront managed prefix list. This can be done before the VPC origin is created as well. For an IPv4 VPC origin, use only `com.amazonaws.global.cloudfront.origin-facing`. For a dual-stack VPC origin, use both `com.amazonaws.global.cloudfront.origin-facing` and `com.amazonaws.global.ipv6.cloudfront.origin-facing`.
- **Option 2:** Allow traffic from the CloudFront service-managed security group (`CloudFront-VPCOrigins-Service-SG`). This can be done only after the VPC origin is created and the service-managed security group is created. This configuration is further restrictive as it restricts the traffic only to your CloudFront distributions.

> **Important**
>
> Do not create your own security group with a name starting with `CloudFront-VPCOrigins-Service-SG`. This is an AWS reserved naming pattern for service-managed security groups. For more information, see Creating a security group.

Protocol and Feature Restrictions

VPC origins do not support the following:

- gRPC traffic
- Origin request and origin response triggers with Lambda@Edge
- Network Access Control Lists (NACLs) – Inbound NACL rules are not evaluated for traffic from CloudFront to your VPC origin. However, outbound NACL rules are evaluated on the return path and must allow traffic on ephemeral TCP ports (1024–65535) to `0.0.0.0/0` or to the CloudFront origin-facing IP ranges.

Create a VPC origin (new distribution)

The following procedure shows you how to create a VPC origin for your new CloudFront distribution in the CloudFront console. Alternatively, you can use the CreateVpcOrigin and CreateDistribution API operations with the AWS CLI or an AWS SDK.

To create a VPC origin for a new CloudFront distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **VPC origins**, **Create VPC origin**.
3. Fill out the required fields. For **Origin ARN**, select the ARN of your Application Load Balancer, Network Load Balancer, or EC2 instance. If you don't see the ARN, you can copy your specific resource ARN and paste it here instead.
4. Select the **Origin type**. The default is IPv4.
5. Choose **Create VPC origin**.
6. Wait for your VPC origin status to change to **Deployed**. This can take up to 15 minutes.
7. Choose **Distributions**, **Create distribution**.
8. For **Origin domain**, select your VPC origins resource from the dropdown list.

   If your VPC origin is an EC2 instance, copy and paste the **Private IP DNS name** of the instance into the **Origin domain** field.
9. Finish creating your distribution. For more information, see Create a CloudFront distribution in the console.

Create a VPC origin (existing distribution)

The following procedure shows you how to create a VPC origin for your existing CloudFront distribution in the CloudFront console, which helps to make sure continuous availability of your applications. Alternatively, you can use the CreateVpcOrigin and UpdateDistributionWithStagingConfig API operations with the AWS CLI or an AWS SDK.

Optionally, you could choose to add your VPC origin to your existing distribution without creating a staging distribution.

To create a VPC origin for your existing CloudFront distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **VPC origins**, **Create VPC origin**.

3. Fill out the required fields. For **Origin ARN**, select the ARN of your Application Load Balancer, Network Load Balancer, or EC2 instance. If you don't see the ARN, you can copy your specific resource ARN and paste it here instead.
4. Select the **Origin type**. The default is IPv4.
5. Choose **Create VPC origin**.
6. Wait for your VPC origin status to change to **Deployed**. This can take up to 15 minutes.
7. In the navigation pane, choose **Distributions**.
8. Choose the ID of your distribution.
9. On the **General** tab, under **Continuous deployment**, choose **Create staging distribution**. For more information, see Use CloudFront continuous deployment to safely test CDN configuration changes.
10. Follow the steps in the **Create staging distribution** wizard to create a staging distribution. Include the following steps:

    - For **Origins**, choose **Create origin**.
    - For **Origin domain**, select your VPC origins resource from the dropdown menu.

      If your VPC origin is an EC2 instance, copy and paste the **Private IP DNS name** of the instance into the **Origin domain** field.
    - Choose **Create origin**.
11. In your staging distribution, test the VPC origin.
12. Promote the staging distribution configuration to your primary distribution. For more information, see Promote a staging distribution configuration.
13. Remove public access to your VPC origin by making the subnet private. After you do this, the VPC origin won't be discoverable over the internet, but CloudFront will still have private access to it. For more information, see Associate or disassociate a subnet with a route table in the *Amazon VPC User Guide*.

Update a VPC origin

The following procedure shows you how to update a VPC origin for your CloudFront distribution in the CloudFront console. Alternatively, you can use the UpdateDistribution and UpdateVpcOrigin API operations with the AWS CLI or an AWS SDK.

To update an existing VPC origin for your CloudFront distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Distributions**.
3. Choose the ID of your distribution.
4. Choose the **Behaviors** tab.
5. Make sure that the VPC origin is not the default origin for your cache behavior.
6. Choose the **Origins** tab.
7. Select the VPC origin that you're going to update and choose **Delete**. This disassociates the VPC origin from your distribution. Repeat steps 2-7 to disassociate the VPC origin from any other distributions.
8. Choose **VPC origins**.
9. Select the VPC origin and choose **Edit**.
10. Make your updates and choose **Update VPC origin**.
11. Wait for your VPC origin status to change to **Deployed**. This can take up to 15 minutes.
12. In the navigation pane, choose **Distributions**.
13. Choose the ID of your distribution.
14. Choose the **Origins** tab.
15. Choose **Create origin**.
16. For **Origin domain**, select your VPC origins resource from the dropdown menu.

    If your VPC origin is an EC2 instance, copy and paste the **Private IP DNS name** of the instance into the **Origin domain** field.
17. Choose **Create origin**. This associates the VPC origin with your distribution again. Repeat steps 12-17 to associate the updated VPC origin with any other distributions.

Supported AWS Regions for VPC origins

VPC origins are currently supported in the following commercial AWS Regions. Availability Zone (AZ) exceptions are noted.

Region Name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1 (except AZ use1-az3)
US West (N. California)	us-west-1 (except AZ usw1-az2)
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Hyderabad)	ap-south-2
Asia Pacific (Jakarta)	ap-southeast-3
Asia Pacific (Melbourne)	ap-southeast-4
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1 (except AZ apne1-az3)
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Thailand)	ap-southeast-7
Asia Pacific (Malaysia)	ap-southeast-5
Asia Pacific (New Zealand)	ap-southeast-6
Asia Pacific (Taipei)	ap-east-2

Region Name	Region
Canada (Central)	ca-central-1 (except AZ cac1-az3)
Canada West (Calgary)	ca-west-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Milan)	eu-south-1
Europe (Paris)	eu-west-3
Europe (Spain)	eu-south-2
Europe (Stockholm)	eu-north-1
Europe (Zurich)	eu-central-2
Israel (Tel Aviv)	il-central-1
Middle East (Bahrain)	me-south-1
Middle East (UAE)	me-central-1
South America (São Paulo)	sa-east-1
Mexico (Central)	mx-central-1

Restrict access to an Amazon S3 Multi-Region Access Point origin

You can use origin access control (OAC) to restrict access to an Amazon S3 Multi-Region Access Point origin. S3 Multi-Region Access Points provide a global endpoint that routes requests to the closest S3 bucket based on network latency.

> **Note**
>
> OAC with S3 Multi-Region Access Points is not supported for MRAP endpoints backed by buckets in opt-in Regions. Requests routed to a bucket in an opt-in Region will fail. All buckets associated with the Multi-Region Access Point must be in Regions that are enabled by default.

For information about using OAC with a standard Amazon S3 bucket origin, see the section called "Restrict access to an Amazon S3 origin".

Prerequisites

Before you create and set up OAC, you must have a CloudFront distribution with an Amazon S3 Multi-Region Access Point origin. The origin domain name must use the S3 Multi-Region Access Point hostname format:

```
multi-region-access-point-alias.accesspoint.s3-global.amazonaws.com
```

For more information about creating an S3 Multi-Region Access Point, see Creating Multi-Region Access Points in the *Amazon Simple Storage Service User Guide*.

Grant CloudFront permission to access the S3 Multi-Region Access Point

Update the Multi-Region Access Point policy to allow the CloudFront service principal (`cloudfront.amazonaws.com`) to access the Multi-Region Access Point. Use a `Condition` element in the policy to allow CloudFront to access the Multi-Region Access Point only when the request is on behalf of the CloudFront distribution that contains the origin.

For information about adding or modifying a Multi-Region Access Point policy, see Multi-Region Access Point policy examples in the *Amazon Simple Storage Service User Guide*.

Example Multi-Region Access Point policy for CloudFront OAC

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontOACAccess",
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3::111122223333:accesspoint/Multi-Region-Access-
Point-Alias.mrap/object/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn":
 "arn:aws:cloudfront::111122223333:distribution/CloudFront distribution ID"
                }
            }
        }
    ]
}
```

Grant CloudFront permission to access the underlying S3 buckets

In addition to the Multi-Region Access Point policy, you must also grant CloudFront permission to access each of the underlying S3 buckets that are associated with the Multi-Region Access Point. You can do this in one of two ways:

> **Important**
>
> You must add this bucket policy to every S3 bucket that is associated with the Multi-Region Access Point. If any bucket is missing the policy, CloudFront requests routed to that bucket will be denied.

Option 1: Grant access only to CloudFront

Add a bucket policy to each S3 bucket that allows the CloudFront service principal to access the bucket. Use this option when you also need to allow direct access to the bucket from other sources.

Example S3 bucket policy for an underlying bucket

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "AllowCloudFrontOACAccessViaMRAP",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket-us-east-1/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn":
 "arn:aws:cloudfront::111122223333:distribution/CloudFront distribution ID"
                }
            }
        }
    ]
}
```

Option 2: Delegate full bucket access to the Multi-Region Access Point

Grant the Multi-Region Access Point full access to each underlying bucket. With this approach, all access to the bucket is controlled by the Multi-Region Access Point policy, which simplifies access management. We recommend this option for use cases that don't require direct access to the bucket.

Example S3 bucket policy that delegates access to the Multi-Region Access Point

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DelegateAccessToMRAP",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket-us-east-1",
                "arn:aws:s3:::amzn-s3-demo-bucket-us-east-1/*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:DataAccessPointArn":
 "arn:aws:s3::111122223333:accesspoint/Multi-Region-Access-Point-Alias.mrap"
                }
```

```
            }
        }
    ]
}
```

For more information, see Multi-Region Access Point policy example in the *Amazon Simple Storage Service User Guide*.

SSE-KMS

If the objects in the underlying S3 buckets are encrypted using server-side encryption with AWS KMS (SSE-KMS), you must make sure that the CloudFront distribution has permission to use the AWS KMS key. Because S3 Multi-Region Access Points can route requests to buckets in multiple Regions, you must add a statement to the KMS key policy in each Region where an underlying bucket uses SSE-KMS. For information about how to modify a key policy, see Changing a key policy in the *AWS Key Management Service Developer Guide*.

Example KMS key policy statement

The following example shows a KMS key policy statement that allows the CloudFront distribution with OAC to access a KMS key for SSE-KMS.

```
{
    "Sid": "AllowCloudFrontServicePrincipalSSE-KMS",
    "Effect": "Allow",
    "Principal": {
        "Service": "cloudfront.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:Encrypt",
        "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceArn": "arn:aws:cloudfront::111122223333:distribution/CloudFront
 distribution ID"
        }
    }
}
```

Important

You must add this key policy statement to the KMS key in every Region where an underlying S3 bucket uses SSE-KMS encryption.

Create the origin access control

To create an origin access control (OAC), you can use the AWS Management Console, CloudFormation, the AWS CLI, or the CloudFront API.

Note

Lambda@Edge origin triggers (both origin request and origin response) are incompatible with origins that use a SigV4a OAC, whether the origin is accessed directly or through an origin group.

Console

To create an origin access control

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Origin access**.
3. Choose **Create control setting**.
4. On the **Create control setting** form, do the following:
   a. In the **Details** pane, enter a **Name** and (optionally) a **Description** for the origin access control.
   b. In the **Settings** pane, we recommend that you leave the default setting (**Sign requests (recommended)**). For more information, see the section called “Advanced settings for origin access control”.
5. Choose **S3** from the **Origin type** dropdown and select the **Use SigV4a signing protocol** option.
6. Choose **Create**.

   After the OAC is created, make note of the **Name**. You need this in the following procedure.

To add an origin access control to an S3 Multi-Region Access Point origin in a distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose a distribution with an S3 Multi-Region Access Point origin that you want to add the OAC to, then choose the **Origins** tab.
3. Select the S3 Multi-Region Access Point origin that you want to add the OAC to, then choose **Edit**.
4. For **Origin access**, choose **Origin access control settings (recommended)**.
5. From the **Origin access control** dropdown menu, choose the OAC that you want to use.
6. Choose **Save changes**.

The distribution starts deploying to all of the CloudFront edge locations. When an edge location receives the new configuration, it signs all requests that it sends to the S3 Multi-Region Access Point origin.

CLI

Use the **create-origin-access-control** command:

```
aws cloudfront create-origin-access-control \
    --origin-access-control-config '{
        "Name": "my-s3-mrap-oac",
        "Description": "OAC for S3 Multi-Region Access Point",
        "SigningProtocol": "sigv4a",
        "SigningBehavior": "always",
        "OriginAccessControlOriginType": "s3"
    }'
```

CloudFormation

Specify the following values in the `OriginAccessControlConfig`:

- `SigningProtocol`: `sigv4a`
- `SigningBehavior`: `always`, `never`, or `no-override`
- `OriginAccessControlOriginType`: `s3`

Example CloudFormation template

```
Type: AWS::CloudFront::OriginAccessControl
Properties:
  OriginAccessControlConfig:
    Description: An optional description for the origin access control
    Name: my-s3-mrap-oac
    OriginAccessControlOriginType: s3
    SigningBehavior: always
    SigningProtocol: sigv4a
```

Signing behavior

The signing behavior options for S3 Multi-Region Access Point origins are the same as those for regular Amazon S3 bucket origins. For more information, see the section called "Advanced settings for origin access control" in *Restrict access to an Amazon S3 origin*.

Restrict access to Application Load Balancers

You can use both internal and internet-facing Application Load Balancers with Amazon CloudFront. You can use internal Application Load Balancers inside private subnets with CloudFront by using VPC origins. CloudFront VPC origins allow you to serve content from applications hosted in private VPC subnets without exposing them to the public internet. For more information, see Restrict access with VPC origins.

If you are using an internet-facing Application Load Balancer with CloudFront, you can use the following security mitigations to prevent users from directly accessing an Application Load Balancer, and allow access only through CloudFront.

1. Configure CloudFront to add a custom HTTP header to requests that it sends to the Application Load Balancer.
2. Configure the Application Load Balancer to only forward requests that contain the custom HTTP header.
3. Require HTTPS to improve the security of this solution.

CloudFront can also help to reduce latency and even absorb some distributed denial of service (DDoS) attacks.

If your use case requires dual access to web applications from both CloudFront and Application Load Balancer directly over the internet, consider splitting your web application APIs as follows:

- APIs that must go through CloudFront. In this case, consider using a separate private Application Load Balancer as an origin.
- APIs that require access through Application Load Balancer. In this case, you bypass CloudFront.

Alternatively, for a web application or other content that's served by an internet-facing Application Load Balancer in Elastic Load Balancing, CloudFront can cache objects and serve them directly to users (viewers), reducing the load on your Application Load Balancer. An internet-facing load balancer has a publicly resolvable DNS name and routes requests from clients to targets over the internet.

For more information, see the following topics. After you complete these steps, users can only access your Application Load Balancer through CloudFront.

Topics

- Configure CloudFront to add a custom HTTP header to requests
- Configure an Application Load Balancer to only forward requests that contain a specific header
- (Optional) Improve the security of this solution
- (Optional) Limit access to origin by using the AWS-managed prefix list for CloudFront

Configure CloudFront to add a custom HTTP header to requests

You can configure CloudFront to add a custom HTTP header to the requests that it sends to your origin (in this case, an Application Load Balancer).

> **Important**
>
> This use case relies on keeping the custom header name and value secret. If the header name and value are not secret, other HTTP clients could potentially include them in requests that they send directly to the Application Load Balancer. This can cause the Application Load Balancer to behave as though the requests came from CloudFront when they did not. To prevent this, keep the custom header name and value secret.

You can configure CloudFront to add a custom HTTP header to origin requests with the CloudFront console, CloudFormation, or the CloudFront API.

To add a custom HTTP header (CloudFront console)

In the CloudFront console, use the **Origin Custom Headers** setting in **Origin Settings**. Enter the **Header Name** and its **Value**.

> **Note**
>
> In production, use randomly generated header names and values. Treat header names and values as secure credentials, like usernames and passwords.

You can edit the **Origin Custom Headers** setting when you create or edit an origin for an existing CloudFront distribution, and when you create a new distribution. For more information, see Update a distribution and Create a distribution.

To add a custom HTTP header (CloudFormation)

In a CloudFormation template, use the `OriginCustomHeaders` property, as shown in the following example.

> **Note**
>
> The header name and value in this example are just for demonstration. In production, use randomly generated values. Treat the header name and value as a secure credential, like a user name and password.

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  TestDistribution:
    Type: 'AWS::CloudFront::Distribution'
    Properties:
      DistributionConfig:
        Origins:
          - DomainName: app-load-balancer.example.com
            Id: Example-ALB
            CustomOriginConfig:
```

```
            OriginProtocolPolicy: https-only
            OriginSSLProtocols:
              - TLSv1.2
          OriginCustomHeaders:
             - HeaderName: X-Custom-Header
               HeaderValue: random-value-1234567890
        Enabled: 'true'
        DefaultCacheBehavior:
          TargetOriginId: Example-ALB
          ViewerProtocolPolicy: allow-all
          CachePolicyId: 658327ea-f89d-4fab-a63d-7e88639e58f6
        PriceClass: PriceClass_All
        ViewerCertificate:
          CloudFrontDefaultCertificate: 'true'
```

For more information, see the Origin and OriginCustomHeader properties in the *AWS CloudFormation User Guide*.

To add a custom HTTP header (CloudFront API)

In the CloudFront API, use the `CustomHeaders` object inside `Origin`. For more information, see CreateDistribution and UpdateDistribution in the *Amazon CloudFront API Reference*, and the documentation for your SDK or other API client.

There are some header names that you can't specify as origin custom headers. For more information, see Custom headers that CloudFront can't add to origin requests.

Configure an Application Load Balancer to only forward requests that contain a specific header

After you configure CloudFront to add a custom HTTP header to the requests that it sends to your Application Load Balancer (see the previous section), you can configure the load balancer to only forward requests that contain this custom header. You do this by adding a new rule and modifying the default rule in your load balancer's listener.

Prerequisites

To use the following procedures, you need an Application Load Balancer with at least one listener. If you haven't created one yet, see Create an Application Load Balancer in the *User Guide for Application Load Balancers*.

The following procedures modify an HTTPS listener. You can use the same process to modify an HTTP listener.

To update the rules in an Application Load Balancer listener

1. Add a new rule. Use the instructions from Add a rule, with the following modifications:
   - Add the rule to the load balancer that is the origin for your CloudFront distribution.
   - For **Add condition**, choose **Http header**. Specify the HTTP header name and value that you added as an origin custom header in CloudFront.
   - For **Add action**, choose **Forward to**. Choose the target group where you want to forward requests.
2. Edit the default rule in your load balancer's listener. Use the instructions from Edit a rule, with the following modifications:
   - Edit the default rule of the load balancer that is the origin for your CloudFront distribution.
   - Delete the default action, and then for **Add action**, choose **Return fixed response**.
   - For **Response code**, enter **`403`**.
   - For **Response body**, enter **`Access denied`**.

After you complete these steps, your load balancer listener has two rules. One rule forwards requests that contain the HTTP header (requests that come from CloudFront). The other rule sends a fixed response to all other requests (requests that don't come from CloudFront).

You can verify that the solution works by sending a request to your CloudFront distribution and one to your Application Load Balancer. The request to CloudFront returns your web application or content, and the one sent directly to your Application Load Balancer returns a `403` response with the plain text message `Access denied`.

(Optional) Improve the security of this solution

To improve the security of this solution, you can configure your CloudFront distribution to always use HTTPS when sending requests to your Application Load Balancer. Remember, this solution only works if you keep the custom header name and value secret. Using HTTPS can help prevent an eavesdropper from discovering the header name and value. We also recommend rotating the header name and value periodically.

Use HTTPS for origin requests

To configure CloudFront to use HTTPS for origin requests, set the **Origin Protocol Policy** setting to **HTTPS Only**. This setting is available in the CloudFront console, CloudFormation, and the CloudFront API. For more information, see Protocol (custom origins only).

The following also applies when you configure CloudFront to use HTTPS for origin requests:

- You must configure CloudFront to forward the `Host` header to the origin with the origin request policy. You can use the AllViewer managed origin request policy.
- Make sure that your Application Load Balancer has an HTTPS listener (as shown in the preceding section). For more information, see Create an HTTPS listener in the *User Guide for Application Load Balancers*. Using an HTTPS listener requires you to have an SSL/TLS certificate that matches the domain name that's routed to your Application Load Balancer.
- SSL/TLS certificates for CloudFront can only be requested (or imported) in the `us-east-1` AWS Region in AWS Certificate Manager (ACM). Because CloudFront is a global service, it automatically distributes the certificate from the `us-east-1` Region to all Regions associated with your CloudFront distribution.
  - For example, if you have an Application Load Balancer (ALB) in the `ap-southeast-2` Region, you must configure SSL/TLS certificates in both the `ap-southeast-2` Region (for using HTTPS between CloudFront and the ALB origin) and the `us-east-1` Region (for using HTTPS between viewers and CloudFront). Both certificates should match the domain name that is routed to your Application Load Balancer. For more information, see AWS Region for AWS Certificate Manager.
- If the end users (also known as *viewers*, or *clients*) of your web application can use HTTPS, you can also configure CloudFront to prefer (or even require) HTTPS connections from the end users. To do this, use the **Viewer Protocol Policy** setting. You can set it to redirect end users from HTTP to HTTPS, or to reject requests that use HTTP. This setting is available in the CloudFront console, CloudFormation, and the CloudFront API. For more information, see Viewer protocol policy.

Rotate the header name and value

In addition to using HTTPS, we also recommend rotating the header name and value periodically. The high-level steps for doing this are as follows:

1. Configure CloudFront to add an additional custom HTTP header to requests that it sends to the Application Load Balancer.
2. Update the Application Load Balancer listener rule to forward requests that contain this additional custom HTTP header.

3. Configure CloudFront to stop adding the original custom HTTP header to requests that it sends to the Application Load Balancer.
4. Update the Application Load Balancer listener rule to stop forwarding requests that contain the original custom HTTP header.

For more information about accomplishing these steps, see the preceding sections.

(Optional) Limit access to origin by using the AWS-managed prefix list for CloudFront

To further restrict access to your Application Load Balancer, you can configure the security group associated with the Application Load Balancer so that it only accepts traffic from CloudFront when the service is using an AWS-managed prefix list. This prevents traffic that doesn't originate from CloudFront from reaching your Application Load Balancer at the network layer (layer 3) or transport layer (layer 4).

For more information, see the Limit access to your origins using the AWS-managed prefix list for Amazon CloudFront blog post.

Restrict the geographic distribution of your content

You can use *geographic restrictions*, sometimes known as *geo blocking*, to prevent users in specific geographic locations from accessing content that you're distributing through an Amazon CloudFront distribution. To use geographic restrictions, you have two options:

- Use the CloudFront geographic restrictions feature. Use this option to restrict access to all of the files that are associated with a distribution and to restrict access at the country level.
- Use a third-party geolocation service. Use this option to restrict access to a subset of the files that are associated with a distribution or to restrict access at a finer granularity than the country level.

Topics

- Use CloudFront geographic restrictions
- Use a third-party geolocation service

Use CloudFront geographic restrictions

When a user requests your content, CloudFront typically serves the requested content regardless of where the user is located. If you need to prevent users in specific countries from accessing your content, you can use the CloudFront geographic restrictions feature to do one of the following:

- Grant permission to your users to access your content only if they're in one of the approved countries on your allowlist.
- Prevent your users from accessing your content if they're in one of the banned countries on your denylist.

For example, if a request comes from a country where you are not authorized to distribute your content, you can use CloudFront geographic restrictions to block the request.

> **Note**
>
> CloudFront determines the location of your users by using a third-party database. The accuracy of the mapping between IP addresses and countries varies by Region. Based on recent tests, the overall accuracy is 99.8%. If CloudFront can't determine a user's location, CloudFront serves the content that the user has requested.

Here's how geographic restrictions work:

1. Suppose you have rights to distribute your content only in Liechtenstein. You update your CloudFront distribution to add an allowlist that contains only Liechtenstein. (Alternatively, you could add a denylist that contains every country except Liechtenstein.)
2. A user in Monaco requests your content, and DNS routes the request to a CloudFront edge location in Milan, Italy.
3. The edge location in Milan looks up your distribution and determines that the user in Monaco does not have permission to download your content.
4. CloudFront returns an HTTP status code `403 (Forbidden)` to the user.

> **Note**
>
> Geographic restrictions do not apply to certificate validation requests for CloudFront managed certificates. When you use a CloudFront managed certificate, requests to the `/.well-known/pki-validation/` path are excluded from geographic restriction rules. This makes sure that Certificate Authorities can validate your certificate from the multiple geographically distributed network perspectives required for issuance and renewal, regardless of your geographic restriction configuration.
> If you use AWS WAF with your distribution, make sure that your AWS WAF rules (including bot control rules) do not block requests to the `/.well-known/pki-validation/` path. Blocking these requests can prevent certificate validation from completing successfully.

You can optionally configure CloudFront to return a custom error message to the user, and you can specify how long you want CloudFront to cache the error response for the requested file. The default value is 10 seconds. For more information, see Create a custom error page for specific HTTP status codes.

Geographic restrictions apply to an entire distribution. If you need to apply one restriction to part of your content and a different restriction (or no restriction) to another part of your content, you must create separate CloudFront distributions or use a third-party geolocation service.

If you enable CloudFront standard logs (access logs), you can identify the requests that CloudFront rejected by searching for the log entries in which the value of `sc-status` (the HTTP status code) is `403`. However, using only the standard logs, you can't distinguish a request that CloudFront rejected based on the location of the user from a request that CloudFront rejected because the user didn't have permission to access the file for another reason. If you have a third-party geolocation service such as Digital Element or MaxMind, you can identify the location of requests based on the IP address in the `c-ip` (client IP) column in the access logs. For more information about CloudFront standard logs, see Access logs (standard logs).

The following procedure explains how to use the CloudFront console to add geographic restrictions to an existing distribution. For information about how to use the console to create a distribution, see Create a distribution.

To add geographic restrictions to your CloudFront web distribution (console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. In the navigation pane, choose **Distributions**, then choose the distribution that you want to update.
3. Choose the **Security** tab, then choose **Geographic restrictions**.
4. Choose **Edit**.
5. Select **Allow list** to create a list of allowed countries, or **Block list** to create a list of blocked countries.
6. Add the desired countries to the list, then choose **Save changes**.

Use a third-party geolocation service

With the CloudFront geographic restrictions feature, you control distribution of your content at the country level for all files that you're distributing with a given web distribution. If you have a use case for geographic restrictions where the restrictions don't follow country boundaries, or if you want to restrict access to only some of the files that you're serving by a given distribution, you can combine CloudFront with a third-party geolocation service. This provides you with control over your content based not only on country but also based on city, ZIP, or postal code, or even latitude and longitude.

When you're using a third-party geolocation service, we recommend that you use CloudFront signed URLs, with which you can specify an expiration date and time after which the URL is no longer valid. In addition, we recommend that you use an Amazon S3 bucket as your origin because you can then use a CloudFront origin access control to prevent users from accessing your content directly from the origin. For more information about signed URLs and origin access control, see Serve private content with signed URLs and signed cookies.

The following steps explain how to control access to your files by using a third-party geolocation service.

To use a third-party geolocation service to restrict access to files in a CloudFront distribution

1. Get an account with a geolocation service.
2. Upload your content to an Amazon S3 bucket.
3. Configure Amazon CloudFront and Amazon S3 to serve private content. For more information, see Serve private content with signed URLs and signed cookies.
4. Write your web application to do the following:

   - Send the IP address for each user request to the geolocation service.

- Evaluate the return value from the geolocation service to determine whether the user is in a location where you want CloudFront to distribute your content.
- If you want to distribute your content to the user's location, generate a signed URL for your CloudFront content. If you don't want to distribute content to that location, return HTTP status code `403 (Forbidden)` to the user. Alternatively, you can configure CloudFront to return a custom error message. For more information, see the section called "Create a custom error page for specific HTTP status codes".

For more information, refer to the documentation for the geolocation service that you're using.

You can use a web server variable to get the IP addresses of the users who are visiting your website. Note the following caveats:

- If your web server is not connected to the internet through a load balancer, you can use a web server variable to get the remote IP address. However, this IP address isn't always the user's IP address. It can also be the IP address of a proxy server, depending on how the user is connected to the internet.
- If your web server is connected to the internet through a load balancer, a web server variable might contain the IP address of the load balancer, not the IP address of the user. In this configuration, we recommend that you use the last IP address in the `X-Forwarded-For` HTTP header. This header typically contains more than one IP address, most of which are for proxies or load balancers. The last IP address in the list is the one most likely to be associated with the user's geographic location.

If your web server is not connected to a load balancer, we recommend that you use web server variables instead of the `X-Forwarded-For` header to avoid IP address spoofing.

Use field-level encryption to help protect sensitive data

With Amazon CloudFront, you can enforce secure end-to-end connections to origin servers by using HTTPS. Field-level encryption adds an additional layer of security that you can use to protect specific data throughout system processing so that only certain applications can see it.

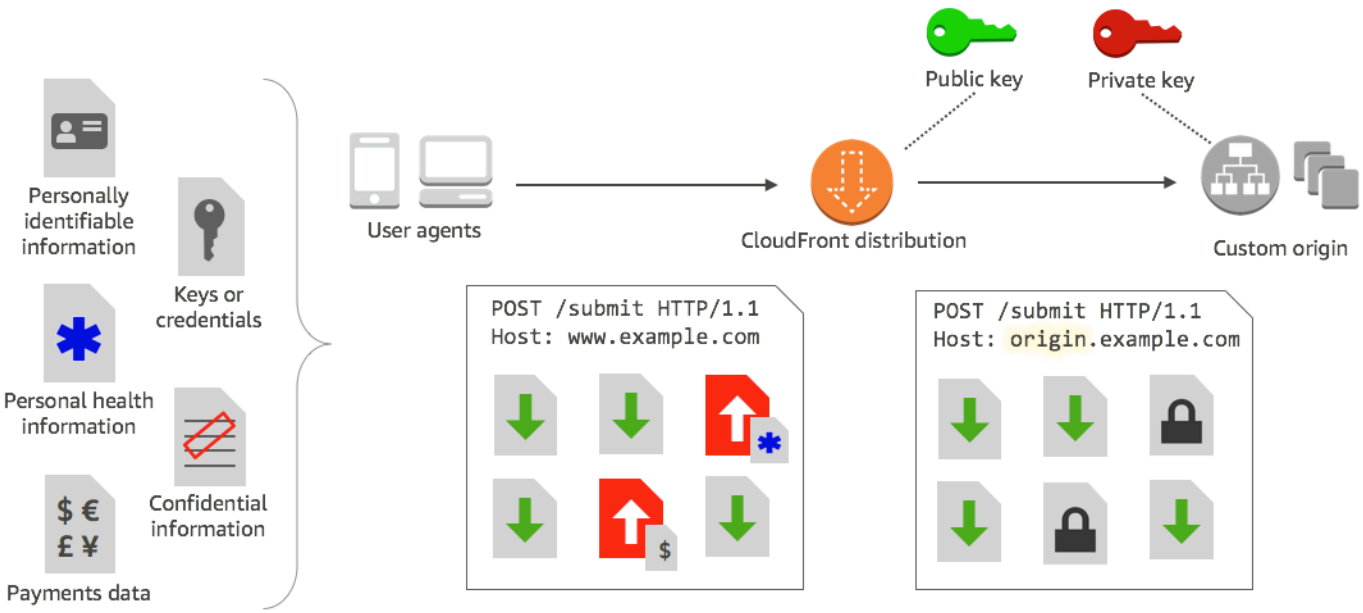
With field-level encryption, you can enable your users to securely upload sensitive information to your web servers. The sensitive information provided by your users is encrypted at the edge, close

to the user, and remains encrypted throughout your entire application stack. This encryption makes sure that only applications that need the data—and have the credentials to decrypt it—are able to do so.

To use field-level encryption, when you configure your CloudFront distribution, specify the set of fields in POST requests that you want to be encrypted, and the public key to use to encrypt them. You can encrypt up to 10 data fields in a request. (You can't encrypt all of the data in a request with field-level encryption; you must specify individual fields to encrypt.)

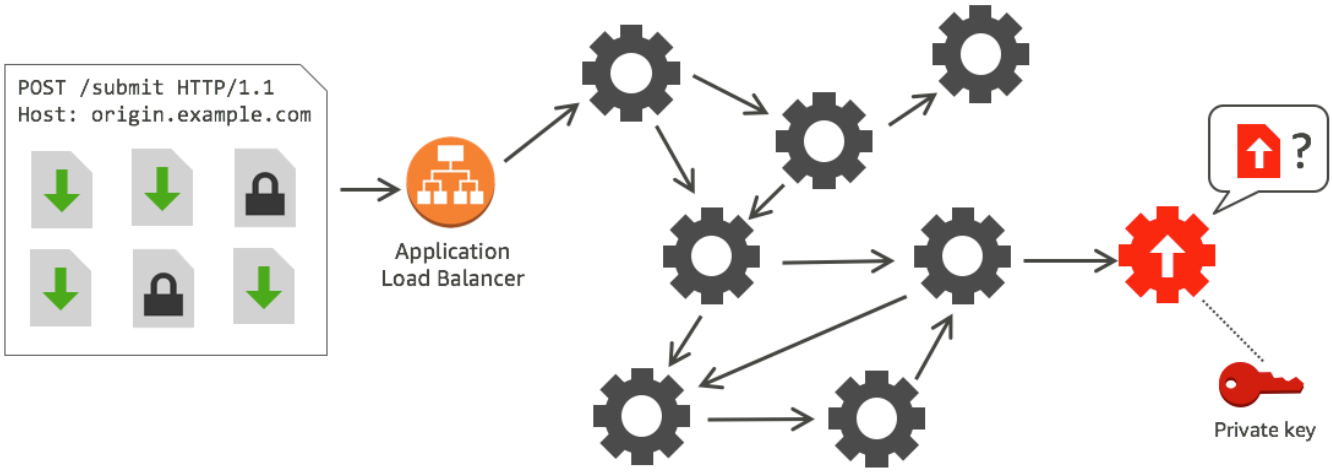
When the HTTPS request with field-level encryption is forwarded to the origin, and the request is routed throughout your origin application or subsystem, the sensitive data is still encrypted, reducing the risk of a data breach or accidental data loss of the sensitive data. Components that need access to the sensitive data for business reasons, such as a payment processing system needing access to a credit card number, can use the appropriate private key to decrypt and access the data.

> **Note**
>
> To use field-level encryption, your origin must support chunked encoding.



CloudFront field-level encryption uses asymmetric encryption, also known as public key encryption. You provide a public key to CloudFront, and all sensitive data that you specify is

encrypted automatically. The key you provide to CloudFront cannot be used to decrypt the encrypted values; only your private key can do that.



Topics

- Overview of field-level encryption
- Set up field-level encryption
- Decrypt data fields at your origin

Overview of field-level encryption

The following steps provide an overview of setting up field-level encryption. For specific steps, see Set up field-level encryption.

1. **Get a public key-private key pair.** You must obtain and add the public key before you start setting up field-level encryption in CloudFront.
2. **Create a field-level encryption profile.** Field-level encryption profiles, which you create in CloudFront, define the fields that you want to be encrypted.
3. **Create a field-level encryption configuration.** A configuration specifies the profiles to use, based on the content type of the request or a query argument, for encrypting specific data fields. You can also choose the request-forwarding behavior options that you want for different scenarios. For example, you can set the behavior for when the profile name specified by the query argument in a request URL doesn't exist in CloudFront.

4. **Link to a cache behavior.** Link the configuration to a cache behavior for a distribution, to specify when CloudFront should encrypt data.

Set up field-level encryption

Follow these steps to get started using field-level encryption. To learn about quotas (formerly known as limits) on field-level encryption, see Quotas.

- Step 1: Create an RSA key pair
- Step 2: Add your public key to CloudFront
- Step 3: Create a profile for field-level encryption
- Step 4: Create a configuration
- Step 5: Add a configuration to a cache behavior

Step 1: Create an RSA key pair

To get started, you must create an RSA key pair that includes a public key and a private key. The public key enables CloudFront to encrypt data, and the private key enables components at your origin to decrypt the fields that have been encrypted. You can use OpenSSL or another tool to create a key pair. The key size must be 2048 bits.

For example, if you're using OpenSSL, you can use the following command to generate a key pair with a length of 2048 bits and save it in the file `private_key.pem`:

```
openssl genrsa -out private_key.pem 2048
```

The resulting file contains both the public and the private key. To extract the public key from that file, run the following command:

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

The public key file (`public_key.pem`) contains the encoded key value that you paste in the following step.

Step 2: Add your public key to CloudFront

After you get your RSA key pair, add your public key to CloudFront.

To add your public key to CloudFront (console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Public key**.
3. Choose **Add public key**.
4. For **Key name**, type a unique name for the key. The name can't have spaces and can include only alphanumeric characters, underscores (_), and hyphens (-). The maximum number of characters is 128.
5. For **Key value**, paste the encoded key value for your public key, including the `-----BEGIN PUBLIC KEY-----` and `-----END PUBLIC KEY-----` lines.
6. For **Comment**, add an optional comment. For example, you could include the expiration date for the public key.
7. Choose **Add**.

You can add more keys to use with CloudFront by repeating the steps in the procedure.

Step 3: Create a profile for field-level encryption

After you add at least one public key to CloudFront, create a profile that tells CloudFront which fields to encrypt.

To create a profile for field-level encryption (console)

1. In the navigation pane, choose **Field-level encryption**.
2. Choose **Create profile**.
3. Fill in the following fields:

   Profile name

   Type a unique name for the profile. The name can't have spaces and can include only alphanumeric characters, underscores (_), and hyphens (-). The maximum number of characters is 128.

   Public key name

   In the drop-down list, choose the name of a public key that you added to CloudFront in step 2. CloudFront uses the key to encrypt the fields that you specify in this profile.

Provider name

Type a phrase to help identify the key, such as the provider where you got the key pair. This information, along with the private key, is needed when applications decrypt data fields. The provider name can't have spaces and can include only alphanumeric characters, colons (:), underscores (_), and hyphens (-). The maximum number of characters is 128.

Field name pattern to match

Type the names of the data fields, or patterns that identify data field names in the request, that you want CloudFront to encrypt. Choose the + option to add all the fields that you want to encrypt with this key.

For the field name pattern, you can type the entire name of the data field, like DateOfBirth, or just the first part of the name with a wildcard character (*), like CreditCard*. The field name pattern must include only alphanumeric characters, square brackets ([and]), periods (.), underscores (_), and hyphens (-), in addition to the optional wildcard character (*).

Make sure that you don’t use overlapping characters for different field name patterns. For example, if you have a field name pattern of ABC*, you can’t add another field name pattern that is AB*. In addition, field names are case-sensitive and the maximum number of characters that you can use is 128.

Comment

(Optional) Type a comment about this profile. The maximum number of characters that you can use is 128.

4. After you fill in the fields, choose **Create profile**.
5. If you want to add more profiles, choose **Add profile**.

Step 4: Create a configuration

After you create one or more field-level encryption profiles, create a configuration that specifies the content type of the request that includes the data to be encrypted, the profile to use for encryption, and other options that specify how you want CloudFront to handle encryption.

For example, when CloudFront can’t encrypt the data, you can specify whether CloudFront should block or forward a request to your origin in the following scenarios:

- **When a request's content type isn't in a configuration** – You might not have added a content type to a configuration. In this case, you can specify whether CloudFront forwards the request with that content type to the origin without encrypting data fields, or blocks the request and returns an error.

  > **Note**
  >
  > If you add a content type to a configuration but haven't specified a profile to use with that type, CloudFront always forwards requests with that content type to the origin.

- **When the profile name provided in a query argument is unknown** – When you specify the `fle-profile` query argument with a profile name that doesn't exist for your distribution, you can specify whether CloudFront should send the request to the origin without encrypting data fields, or block the request and return an error.

In a configuration, you can also specify whether providing a profile as a query argument in a URL overrides a profile that you've mapped to the content type for that query. By default, CloudFront uses the profile that you've mapped to a content type, if you specify one. With this, you can have a profile that's used by default but decide for certain requests that you want to enforce a different profile.

So, for example, you might specify (in your configuration) **`SampleProfile`** as the query argument profile to use. Then you could use the URL `https://d1234.cloudfront.net?fle-profile=SampleProfile` instead of `https://d1234.cloudfront.net`, to have CloudFront use **`SampleProfile`** for this request, instead of the profile you'd set up for the content type of the request.

You can create up to 10 configurations for a single account, and then associate one of the configurations to the cache behavior of any distribution for the account.

To create a configuration for field-level encryption (console)

1. On the **Field-level encryption** page, choose **Create configuration**.

   Note: If you haven't created at least one profile, you won't see the option to create a configuration.

2. Fill in the following fields to specify the profile to use. (Some fields can't be changed.)

Content type (can't be changed)

The content type is set to `application/x-www-form-urlencoded` and can't be changed.

Default profile ID (optional)

In the drop-down list, choose the profile that you want to map to the content type in the **Content type** field.

Content format (can't be changed)

The content format is set to `URLencoded` and can't be changed.

3. If you want to change the CloudFront default behavior for the following options, select the appropriate check box.

   Forward request to origin when request's content type is not configured

   Select the check box if you want to allow the request to go to your origin *if you have not specified a profile to use for the content type of the request*.

   Override the profile for a content type with a provided query argument

   Select the check box if you want to allow a profile provided in a query argument to *override the profile that you've specified for a content type*.

4. If you select the check box to allow a query argument to override the default profile, you must complete the following additional fields for the configuration. You can create up to five of these query argument mappings to use with queries.

   Query argument

   Type the value that you want to include in URLs for the `fle-profile` query argument. This value tells CloudFront to use the profile ID (that you specify in the next field) associated with this query argument for field-level encryption for this query.

   The maximum number of characters that you can use is 128. The value can't include spaces, and must use only alphanumeric or the following characters: dash (-), period (.), underscore (_), asterisk (*), plus-sign (+), percent (%).

Profile ID

In the drop-down list, choose the profile that you want to associate with the value that you typed for **Query argument**.

Forward request to origin when the profile specified in a query argument does not exist

Select the check box if you want to allow the request to go to your origin *if the profile specified in a query argument isn't defined in CloudFront*.

Step 5: Add a configuration to a cache behavior

To use field-level encryption, link a configuration to a cache behavior for a distribution by adding the configuration ID as a value for your distribution.

> **Important**
>
> To link a field-level encryption configuration to a cache behavior, the distribution must be configured to always use HTTPS, and to accept HTTP `POST` and `PUT` requests from viewers. That is, the following must be true:
>
> - The cache behavior's **Viewer Protocol Policy** must be set to **Redirect HTTP to HTTPS** or **HTTPS Only**. (In CloudFormation or the CloudFront API, `ViewerProtocolPolicy` must be set to `redirect-to-https` or `https-only`.)
> - The cache behavior's **Allowed HTTP Methods** must be set to **GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE**. (In CloudFormation or the CloudFront API, `AllowedMethods` must be set to `GET`, `HEAD`, `OPTIONS`, `PUT`, `POST`, `PATCH`, `DELETE`. These can be specified in any order.)
> - The origin setting's **Origin Protocol Policy** must be set to **Match Viewer** or **HTTPS Only**. (In CloudFormation or the CloudFront API, `OriginProtocolPolicy` must be set to `match-viewer` or `https-only`.)

For more information, see All distribution settings reference.

Decrypt data fields at your origin

CloudFront encrypts data fields by using the AWS Encryption SDK. The data remains encrypted throughout your application stack and can be accessed only by applications that have the credentials to decrypt it.

After encryption, the ciphertext is base64 encoded. When your applications decrypt the text at the origin, they must first decode the ciphertext, and then use the AWS Encryption SDK to decrypt the data.

The following code example illustrates how applications can decrypt data at your origin. Note the following:

- To simplify the example, this sample loads public and private keys (in DER format) from files in the working directory. In practice, you would store the private key in a secure offline location, such as an offline hardware security module, and distribute the public key to your development team.
- CloudFront uses specific information while encrypting the data, and the same set of parameters should be used at the origin to decrypt it. Parameters CloudFront uses while initializing the MasterKey include the following:
  - PROVIDER_NAME: You specified this value when you created a field-level encryption profile. Use the same value here.
  - KEY_NAME: You created a name for your public key when you uploaded it to CloudFront, and then specified the key name in the profile. Use the same value here.
  - ALGORITHM: CloudFront uses `RSA/ECB/OAEPWithSHA-256AndMGF1Padding` as the algorithm for encrypting, so you must use the same algorithm to decrypt the data.
- If you run the following sample program with ciphertext as input, the decrypted data is output to your console. For more information, see the Java Example Code in the AWS Encryption SDK.

Sample code

```
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
```

```
import java.security.spec.X509EncodedKeySpec;

import org.apache.commons.codec.binary.Base64;

import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoResult;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;

/**
 * Sample example of decrypting data that has been encrypted by CloudFront field-level
 encryption.
 */
public class DecryptExample {

    private static final String PRIVATE_KEY_FILENAME = "private_key.der";
    private static final String PUBLIC_KEY_FILENAME = "public_key.der";
    private static PublicKey publicKey;
    private static PrivateKey privateKey;

    // CloudFront uses the following values to encrypt data, and your origin must use
 same values to decrypt it.
    // In your own code, for PROVIDER_NAME, use the provider name that you specified
 when you created your field-level
    // encryption profile. This sample uses 'DEMO' for the value.
    private static final String PROVIDER_NAME = "DEMO";
    // In your own code, use the key name that you specified when you added your public
 key to CloudFront. This sample
    // uses 'DEMOKEY' for the key name.
    private static final String KEY_NAME = "DEMOKEY";
    // CloudFront uses this algorithm when encrypting data.
    private static final String ALGORITHM = "RSA/ECB/OAEPWithSHA-256AndMGF1Padding";

    public static void main(final String[] args) throws Exception {

        final String dataToDecrypt = args[0];

        // This sample uses files to get public and private keys.
        // In practice, you should distribute the public key and save the private key
 in secure storage.
        populateKeyPair();

        System.out.println(decrypt(debase64(dataToDecrypt)));
    }
```

```
    private static String decrypt(final byte[] bytesToDecrypt) throws Exception {
        // You can decrypt the stream only by using the private key.

        // 1. Instantiate the SDK
        final AwsCrypto crypto = new AwsCrypto();

        // 2. Instantiate a JCE master key
        final JceMasterKey masterKey = JceMasterKey.getInstance(
                publicKey,
                privateKey,
                PROVIDER_NAME,
                KEY_NAME,
                ALGORITHM);

        // 3. Decrypt the data
        final CryptoResult <byte[], ? > result = crypto.decryptData(masterKey,
 bytesToDecrypt);
        return new String(result.getResult());
    }

    // Function to decode base64 cipher text.
    private static byte[] debase64(final String value) {
        return Base64.decodeBase64(value.getBytes());
    }

    private static void populateKeyPair() throws Exception {
        final byte[] PublicKeyBytes =
 Files.readAllBytes(Paths.get(PUBLIC_KEY_FILENAME));
        final byte[] privateKeyBytes =
 Files.readAllBytes(Paths.get(PRIVATE_KEY_FILENAME));
        publicKey = KeyFactory.getInstance("RSA").generatePublic(new
 X509EncodedKeySpec(PublicKeyBytes));
        privateKey = KeyFactory.getInstance("RSA").generatePrivate(new
 PKCS8EncodedKeySpec(privateKeyBytes));
    }
}
```

Video on demand and live streaming video with CloudFront

You can use CloudFront to deliver video on demand (VOD) or live streaming video by using any HTTP origin. One way you can set up video workflows in the cloud is by using CloudFront together with AWS Media Services.

Topics

- About streaming video
- Deliver video on demand with CloudFront
- Deliver video streaming with CloudFront and AWS Media Services
- Media quality-aware resiliency

About streaming video

You must use an encoder to package video content before CloudFront can distribute the content. The packaging process creates *segments* that contain your audio, video, and captions content. It also generates manifest files, which describe in a specific order what segments to play and when. Common package formats are MPEG DASH, Apple HLS, Microsoft Smooth Streaming, and CMAF.

VOD streaming

For VOD streaming, your video content is stored on a server and viewers can watch it at any time. To make an asset that viewers can stream, use an encoder, such as AWS Elemental MediaConvert, to format and package your media files.

After your video is packaged into the right formats, you can store it on a server or in an Amazon S3 bucket, and then deliver it with CloudFront as viewers request it.

Live video streaming

For live video streaming, your video content is streamed real time as live events happen, or is set up as a 24x7 live channel. To create live outputs for broadcast and streaming delivery, use an encoder such as AWS Elemental MediaLive, to compress the video and format it for viewing devices.

After your video is encoded, you can store it in AWS Elemental MediaStore or convert it into different delivery formats by using AWS Elemental MediaPackage. Use either of these origins to set up a CloudFront distribution to deliver the content. For specific steps and guidance for creating distributions that work together with these services, see Serve video by using AWS Elemental MediaStore as the origin and Serve live video formatted with AWS Elemental MediaPackage.

Wowza and Unified Streaming also provide tools that you can use for streaming video with CloudFront. For more information about using Wowza with CloudFront, see Bring your Wowza Streaming Engine license to CloudFront live HTTP streaming on the Wowza documentation website. For information about using Unified Streaming with CloudFront for VOD streaming, see CloudFront on the Unified Streaming documentation website.

Deliver video on demand with CloudFront

To deliver video on demand (VOD) streaming with CloudFront, use the following services:

- Amazon S3 to store the content in its original format and to store the transcoded video.
- An encoder (such as AWS Elemental MediaConvert) to transcode the video into streaming formats.
- CloudFront to deliver the transcoded video to viewers. For Microsoft Smooth Streaming, see Configure video on demand for Microsoft Smooth Streaming.

To create a VOD solution with CloudFront

1. Upload your content to an Amazon S3 bucket. To learn more about working with Amazon S3, see the Amazon Simple Storage Service User Guide.
2. Transcode your content by using a MediaConvert job. The job converts your video into the formats required by the players that your viewers use. You can also use the job to create assets that vary in resolution and bitrate. These assets are used for adaptive bitrate (ABR) streaming, which adjusts the viewing quality depending on the viewer's available bandwidth. MediaConvert stores the transcoded video in an S3 bucket.
3. Deliver your converted content by using a CloudFront distribution. Viewers can watch the content on any device, at any time.

Configure video on demand for Microsoft Smooth Streaming

You have the following options for using CloudFront to distribute video on demand (VOD) content that you've transcoded into the Microsoft Smooth Streaming format:

- Specify a web server that runs Microsoft IIS and supports Smooth Streaming as the origin for your distribution.
- Enable Smooth Streaming in the cache behaviors of a CloudFront distribution. Because you can use multiple cache behaviors in a distribution, you can use one distribution for Smooth Streaming media files as well as other content.

> **Important**
>
> If you specify a web server running Microsoft IIS as your origin, do *not* enable Smooth Streaming in the cache behaviors of your CloudFront distribution. CloudFront can't use a Microsoft IIS server as an origin if you enable Smooth Streaming as a cache behavior.

If you enable Smooth Streaming in a cache behavior (that is, you do not have a server that is running Microsoft IIS), note the following:

- You can still distribute other content using the same cache behavior if the content matches the value of **Path Pattern** for that cache behavior.
- CloudFront can use either an Amazon S3 bucket or a custom origin for Smooth Streaming media files. CloudFront cannot use a Microsoft IIS Server as an origin if you enable Smooth Streaming for the cache behavior.
- You cannot invalidate media files in the Smooth Streaming format. If you want to update files before they expire, you must rename them. For more information, see Add, remove, or replace content that CloudFront distributes.

For information about Smooth Streaming clients, see Smooth Streaming on the Microsoft documentation website.

To use CloudFront to distribute Smooth Streaming files when a Microsoft IIS web server isn't the origin

1. Transcode your media files into Smooth Streaming fragmented MP4 format.

2. Do one of the following:

   - **If you're using the CloudFront console:** When you create or update a distribution, enable Smooth Streaming in one or more of the distribution's cache behaviors.
   - **If you're using the CloudFront API:** Add the `SmoothStreaming` element to the `DistributionConfig` complex type for one or more of the distribution's cache behaviors.

3. Upload the Smooth Streaming files to your origin.

4. Create either a `clientaccesspolicy.xml` or a `crossdomainpolicy.xml` file, and add it to a location that is accessible at the root of your distribution, for example, `https://d111111abcdef8.cloudfront.net/clientaccesspolicy.xml`. The following is an example policy:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <access-policy>
   <cross-domain-access>
   <policy>
   <allow-from http-request-headers="*">
   <domain uri="*"/>
   </allow-from>
   <grant-to>
   <resource path="/" include-subpaths="true"/>
   </grant-to>
   </policy>
   </cross-domain-access>
   </access-policy>
   ```

   For more information, see Making a Service Available Across Domain Boundaries on the Microsoft Developer Network website.

5. For links in your application (for example, a media player), specify the URL for the media file in the following format:

   `https://d111111abcdef8.cloudfront.net/video/presentation.ism/Manifest`

Deliver video streaming with CloudFront and AWS Media Services

To use AWS Media Services with CloudFront to deliver live content to a global audience, see the following guidance.

Use AWS Elemental MediaLive to encode live video streams in real time. To encode a large video stream, MediaLive compresses it into smaller versions (*encodes*) that can be distributed to your viewers.

After you compress a live video stream, you can use either of the following two main options to prepare and serve the content:

- **Convert your content into required formats, and then serve it** – If you require content in multiple formats, use AWS Elemental MediaPackage to package the content for different device types. When you package the content, you can also implement extra features and add digital rights management (DRM) to prevent unauthorized use of your content. For step-by-step instructions for using CloudFront to serve content that MediaPackage formatted, see Serve live video formatted with AWS Elemental MediaPackage.
- **Store and serve your content using scalable origin** – If MediaLive encoded content in the formats required by all of the devices that your viewers use, use a highly scalable origin like AWS Elemental MediaStore to serve the content. For step-by-step instructions for using CloudFront to serve content that is stored in a MediaStore container, see Serve video by using AWS Elemental MediaStore as the origin.

After you've set up your origin by using one of these options, you can distribute live streaming video to viewers by using CloudFront.

> **Tip**
>
> You can learn about an AWS solution that automatically deploys services for building a highly available real-time viewing experience. To see the steps to automatically deploy this solution, see Live Streaming Automated Deployment.

Topics

- Serve video by using AWS Elemental MediaStore as the origin

- Serve live video formatted with AWS Elemental MediaPackage
- Serve video-on-demand content with AWS Elemental MediaPackage

Serve video by using AWS Elemental MediaStore as the origin

If you have video stored in an AWS Elemental MediaStore container, you can create a CloudFront distribution to serve the content.

To get started, you grant CloudFront access to your MediaStore container. Then you create a CloudFront distribution and configure it to work with MediaStore.

To serve content from an AWS Elemental MediaStore container

1. Follow the procedure at Allowing Amazon CloudFront to access your AWS Elemental MediaStore container, and then return to these steps to create your distribution.
2. Create a distribution with the following settings:

    a. **Origin domain** – The data endpoint that is assigned to your MediaStore container. From the dropdown list, choose the MediaStore container for your live video.

    b. **Origin path** – The folder structure in the MediaStore container where your objects are stored. For more information, see the section called “Origin path”.

    c. **Add custom header** – Add header names and values if you want CloudFront to add custom headers when it forwards requests to your origin.

    d. **Viewer protocol policy** – Choose **Redirect HTTP to HTTPS**. For more information, see the section called “Viewer protocol policy”.

    e. **Cache policy** and **Origin request policy**

    - For **Cache policy**, choose **Create policy**, and then create a cache policy that’s appropriate for your caching needs and segment durations. After you create the policy, refresh the list of cache policies and choose the policy that you just created.
    - For **Origin request policy**, choose **CORS-CustomOrigin** from the dropdown list.

    For the other settings, you can set specific values based on other technical requirements or the needs of your business. For a list of all the options for distributions and information about setting them, see the section called “All distribution settings”.

3. For links in your application (for example, a media player), specify the name of the media file in the same format that you use for other objects that you're distributing using CloudFront.

Serve live video formatted with AWS Elemental MediaPackage

If you formatted a live stream by using AWS Elemental MediaPackage, you can create a CloudFront distribution and configure cache behaviors to serve the live stream. The following process assumes that you have already created a channel and added endpoints for your live video using MediaPackage.

To create a CloudFront distribution for MediaPackage manually, follow these steps:

Step 1: Create and configure a CloudFront distribution

Complete the following procedure to set up a CloudFront distribution for the live video channel that you created with MediaPackage.

To create a distribution for your live video channel

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **Create distribution**.
3. Choose the settings for the distribution, including the following:

   Origin domain

   The origin where your MediaPackage live video channel and endpoints are. Choose the text field, then from the dropdown list, choose the MediaPackage origin domain for your live video. You can map one domain to several origin endpoints.

   If you created your origin domain using another AWS account, type the origin URL value into the field. The origin must be an HTTPS URL.

   For example, for an HLS endpoint like `https://3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com/out/v1/abc123/index.m3u8`, the origin domain is `3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com`.

   For more information, see the section called "Origin domain".

Origin path

The path to the MediaPackage endpoint from where the content is served.

For more information about how an origin path works, see the section called “Origin path”.

> **Important**
>
> The wildcard path `*` is required to route somewhere in the CloudFront distribution. To prevent requests not matching an explicit path from routing to the real origin, create a "dummy" origin for that wildcard path.

Example: Creating a "dummy" origin

In the following example, the endpoints `abc123` and `def456` route to the "real" origin, but requests for any other endpoint's video content route to `mediapackage.us-west-2.amazonaws.com` without the proper subdomain, which results in an HTTP `404` error.

MediaPackage endpoints:

```
https://3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com/out/v1/abc123/
index.m3u8
https://3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com/out/v1/def456/
index.m3u8
```

CloudFront Origin A:

```
Domain: 3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com
Path: None
```

CloudFront Origin B:

```
Domain: mediapackage.us-west-2.amazonaws.com
Path: None
```

CloudFront cache behavior:

```
1. Path: /out/v1/abc123/* forward to Origin A
2. Path: /out/v1/def456/* forward to Origin A
3. Path: * forward to Origin B
```

For the other distribution settings, set specific values based on other technical requirements or the needs of your business. For a list of all the options for distributions and information about setting them, see the section called “All distribution settings”.

When you finish choosing the other distribution settings, choose **Create distribution**.

4. Choose the distribution that you just created, then choose **Behaviors**.
5. Select the default cache behavior, then choose **Edit**. Specify the correct cache behavior settings for the channel that you chose for the origin. Later, you’ll add one or more additional origins and edit cache behavior settings for them.
6. Go to the CloudFront distributions page.
7. Wait until the value of the **Last modified** column for your distribution has changed from **Deploying** to a date and time, indicating that CloudFront has created your distribution.

Step 2: Add Origins for the domains of your MediaPackage endpoints

Repeat the steps here to add each of your MediaPackage channel endpoints to your distribution, keeping in mind the need to create a "dummy" origin.

To add other endpoints as origins

1. On the CloudFront console, choose the distribution that you created for your channel.
2. Choose **Origins**, then choose **Create origin**.
3. For **Origin domain**, in the dropdown list, choose a MediaPackage endpoint for your channel.
4. For the other settings, set the values based on other technical requirements or the needs of your business. For more information, see the section called “Origin settings”.
5. Choose **Create origin**.

Step 3: Configure cache behaviors for all endpoints

For each endpoint, you must configure cache behaviors to add path patterns that route requests correctly. The path patterns that you specify depend on the video format that you’re serving. The

following procedure includes the path pattern information to use for Apple HLS, CMAF, DASH, and Microsoft Smooth Streaming formats.

You typically set up two cache behaviors for each endpoint:

- The parent manifest, which is the index to your files.
- The segments, which are the files of the video content.

To create a cache behavior for an endpoint

1. On the CloudFront console, choose the distribution that you created for your channel.
2. Choose **Behaviors**, then choose **Create behavior**.
3. For **Path pattern**, use a specific MediaPackage `OriginEndpoint` GUID as a path prefix.

   Path patterns

   For an HLS endpoint like `https://3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com/out/v1/abc123/index.m3u8`, create the following two cache behaviors:

   - For parent and child manifests, use `/out/v1/abc123/*.m3u8`.
   - For the content segments, use `/out/v1/abc123/*.ts`.

   For a CMAF endpoint like `https://3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com/out/v1/abc123/index.m3u8`, create the following two cache behaviors:

   - For parent and child manifests, use `/out/v1/abc123/*.m3u8`.
   - For the content segments, use `/out/v1/abc123/*.mp4`.

   For a DASH endpoint like `https://3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com/out/v1/abc123/index.mpd`, create the following two cache behaviors:

   - For the parent manifest, use `/out/v1/abc123/*.mpd`.
   - For the content segments, use `/out/v1/abc123/*.mp4`.

   For a Microsoft Smooth Streaming endpoint like `https://3ae97e9482b0d011.mediapackage.us-west-2.amazonaws.com/out/`

`v1/abc123/index.ism`, only a manifest is served, so you create only one cache behavior: `out/v1/abc123/index.ism/*`.

4. For each cache behavior, specify values for the following settings:

   Viewer protocol policy

   Choose **Redirect HTTP to HTTPS**.

   Cache policy and origin request policy

   For **Cache policy**, choose **Create policy**. For your new cache policy, specify the following settings:

   Minimum TTL

   Set to 5 seconds or less, to help prevent serving stale content.

   Query strings

   For **Query strings** (in **Cache key settings**), choose **Include specified query strings**. For **Allow**, add the following values by typing them and then choosing **Add item**:

   - Add `m` as a query string parameter that you want CloudFront to use as the basis for caching. The MediaPackage response always includes the tag `?m=###` to capture the modified time of the endpoint. If content is already cached with a different value for this tag, CloudFront requests a new manifest instead of serving the cached version.
   - If you're using the time-shifted viewing functionality in MediaPackage, specify `start` and `end` as additional query string parameters on the cache behavior for manifest requests (`*.m3u8`, `*.mpd`, and `index.ism/*`). This way, content is served that's specific to the requested time period in the manifest request. For more information about time-shifted viewing and formatting content start and end request parameters, see Time-shifted viewing in the *AWS Elemental MediaPackage User Guide*.
   - If you're using the manifest filtering feature in MediaPackage, specify `aws.manifestfilter` as an additional query string parameter for the cache policy that you use with the cache behavior for manifest requests (`*.m3u8`, `*.mpd`, and `index.ism/*`). This configures your distribution to forward the `aws.manifestfilter` query string to your MediaPackage origin, which is required for the manifest filtering feature to work. For more information, see Manifest filtering in the *AWS Elemental MediaPackage User Guide*.

- If you're using low-latency HLS (LL-HLS), specify `_HLS_msn` and `_HLS_part` as additional query string parameters for the cache policy that you use with the cache behavior for manifest requests (`*.m3u8`). This configures your distribution to forward the `_HLS_msn` and `_HLS_part` query strings to your MediaPackage origin, which is required for the LL-HLS blocking playlist request feature to work.

5. Choose **Create**.
6. After you create the cache policy, go back to the cache behavior creation workflow. Refresh the list of cache policies, and choose the policy that you just created.
7. Choose **Create behavior**.
8. If your endpoint is not a Microsoft Smooth Streaming endpoint, repeat these steps to create a second cache behavior.

Step 4: Enable header-based MediaPackage CDN Authorization

We recommend enabling header-based MediaPackage CDN Authorization between MediaPackage endpoints and the CloudFront distribution. For more information, see Enable CDN authorization in MediaPackage in the *AWS Elemental MediaPackage User Guide*.

Step 5: Use CloudFront to serve the live stream channel

After you create the distribution, add the origins, create the cache behaviors, and enable header-based CDN authorization, you can serve the live stream channel using CloudFront. CloudFront routes requests from viewers to the correct MediaPackage endpoints based on the settings that you configured for the cache behaviors.

For links in your application (for example, a media player), specify the URL for the media file in the standard format for CloudFront URLs. For more information, see the section called "Customize file URLs".

Serve video-on-demand content with AWS Elemental MediaPackage

If you originate your video-on-demand (VOD) content from an AWS Elemental MediaPackage origin, you can create a CloudFront distribution and configure optimized cache behaviors to serve the VOD content to viewers. The following process assumes that you have already created a packaging group with a packaging configuration and ingested an asset with MediaPackage.

To create a CloudFront distribution for MediaPackage manually, follow these steps:

Step 1: Create and configure a CloudFront distribution

Complete the following procedure to set up a CloudFront distribution for the packaging group that you created with MediaPackage.

To create a distribution for your VOD content

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose **Create distribution**.
3. Choose the settings for the distribution, including the following:

   Origin domain

   The origin for your MediaPackage packaging group. Type the origin URL value into the text field. The origin must be an HTTPS URL.

   For example, for an HLS endpoint like `https://3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com/out/v1/abc123/def456/ghi789/index.m3u8`, the origin domain is `3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com`.

   For more information, see the section called “Origin domain”.

   Origin path

   The path from where the content is served.

   For more information about how an origin path works, see the section called “Origin path”.

> **Important**
>
> The wildcard path * is required to route somewhere in the CloudFront distribution. To prevent requests not matching an explicit path from routing to the real origin, create a "dummy" origin for that wildcard path.

Example: Creating a "dummy" origin

In the following example, the packaging configurations `def456` and `321xyz` route to the "real" origin, but requests for any other video content route to `mediapackage-vod.us-west-2.amazonaws.com` without the proper subdomain, which results in an HTTP `404` error.

MediaPackage content URLs for a single asset for a packaging group with two packaging configurations:

```
https://3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com/out/v1/
abc123/def456/ghi789/index.m3u8
https://3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com/out/v1/
abc123/321xyz/654uvw/index.m3u8
```

CloudFront Origin A:

```
Domain: 3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com
Path: None
```

CloudFront Origin B:

```
Domain: mediapackage-vod.us-west-2.amazonaws.com
Path: None
```

CloudFront cache behavior:

```
1. Path: /out/v1/*/def456/* forward to Origin A
2. Path: /out/v1/*/321xyz/* forward to Origin A
3. Path: * forward to Origin B
```

For the other distribution settings, set specific values based on other technical requirements or the needs of your business. For a list of all the options for distributions and information about setting them, see the section called “All distribution settings”.

When you finish choosing the other distribution settings, choose **Create distribution**.

4. Choose the distribution that you just created, then choose **Behaviors**.

5. Select the default cache behavior, then choose **Edit**. Specify the correct cache behavior settings for the packaging configuration that you chose for the origin. Later, you'll add one or more additional origins and edit cache behavior settings for them.
6. Go to the CloudFront distributions page.
7. Wait until the value of the **Last modified** column for your distribution has changed from **Deploying** to a date and time, indicating that CloudFront has created your distribution.

Step 2: Add Origins for the domains of your MediaPackage packaging groups

Repeat the steps here to add each of your MediaPackage packaging groups to your distribution, keeping in mind the need to create a "dummy" origin.

To add other packaging groups as origins

1. On the CloudFront console, choose the distribution that you created for your channel.
2. Choose **Origins**, then choose **Create origin**.
3. For **Origin domain**, type in the URL for the MediaPackage packaging group.
4. For the other settings, set the values based on other technical requirements or the needs of your business. For more information, see the section called “Origin settings”.
5. Choose **Create origin**.

Step 3: Configure cache behaviors for all packaging configurations

For each packaging configuration, you must configure cache behaviors to add path patterns that route requests correctly. The path patterns that you specify depend on the video format that you’re serving. The following procedure includes the path pattern information to use for Apple HLS, CMAF, DASH, and Microsoft Smooth Streaming formats.

You typically set up multiple cache behaviors for each packaging configuration:

- The parent manifest, which is the index to your files.
- The segments, which are the files of the video content. A format might use more than one extension for content, depending on your configuration. A cache behavior is needed for each extension.

To create a cache behavior for a packaging configuration

1. On the CloudFront console, choose the distribution that you created for your channel.
2. Choose **Behaviors**, then choose **Create behavior**.
3. For **Path pattern**, use a specific MediaPackage VOD packaging configuration GUID as a path prefix. This is the second GUID in a MediaPackage VOD path.

   Path patterns

   For HLS content like `https://3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com/out/v1/abc123/def456/ghi789/index.m3u8`, create the following cache behaviors:

   - For parent and child manifests, use `/out/v1/*/def456/*.m3u8`.
   - For the content segments, use `/out/v1/*/def456/*.ts` and repeat for all needed segment extensions.

   For CMAF content like `https://3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com/out/v1/abc123/def456/ghi789/index.m3u8`, create the following cache behaviors:

   - For parent and child manifests, use `/out/v1/*/def456/*.m3u8`.
   - For the content segments, use `/out/v1/*/def456/*.mp4` and repeat for all needed segment extensions.

   For DASH content like `https://3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com/out/v1/abc123/def456/ghi789/index.mpd`, create the following cache behaviors:

   - For the parent manifest, use `/out/v1/*/def456/*.mpd`.
   - For the content segments, use `/out/v1/*/def456/*.mp4`.

   For a Microsoft Smooth Streaming endpoint like `https://3ae97e9482b0d011.egress.mediapackage-vod.us-west-2.amazonaws.com/out/v1/abc123/def456/ghi789/index.ism/Manifest`, only a manifest is served, so you create only one cache behavior: `out/v1/*/def456/*/index.ism/*`.
4. For each cache behavior, specify values for the following settings:

Viewer protocol policy

Choose **Redirect HTTP to HTTPS**.

Cache policy and origin request policy

For **Cache policy**, choose **Create policy**. For your new cache policy, specify the following settings:

Minimum TTL

Set to 5 seconds or less, to help prevent serving stale content.

Query strings

For **Query strings** (in **Cache key settings**), choose **Include specified query strings**. For **Allow**, add the following values by typing them and then choosing **Add item**:

- If you're using the manifest filtering feature in MediaPackage, specify `aws.manifestfilter` as an additional query string parameter for the cache policy that you use with the cache behavior for manifest requests (`*.m3u8`, `*.mpd`, and `index.ism/*`). This configures your distribution to forward the `aws.manifestfilter` query string to your MediaPackage origin, which is required for the manifest filtering feature to work. For more information, see Manifest filtering in the *AWS Elemental MediaPackage User Guide*.

5. Choose **Create**.
6. After you create the cache policy, go back to the cache behavior creation workflow. Refresh the list of cache policies, and choose the policy that you just created.
7. Choose **Create behavior**.
8. If your endpoint is not a Microsoft Smooth Streaming endpoint, repeat these steps to create a second cache behavior.

Step 4: Enable header-based MediaPackage CDN Authorization

We recommend enabling header-based MediaPackage CDN Authorization between MediaPackage VOD content and the CloudFront distribution. For more information, see Enable CDN authorization in MediaPackage in the *AWS Elemental MediaPackage User Guide*.

Step 5: Use CloudFront to serve the VOD content

After you create the distribution, add the origins, create the cache behaviors, and enable header-based CDN authorization, you can serve the VOD content using CloudFront. CloudFront routes requests from viewers to the correct MediaPackage VOD content based on the settings that you configured for the cache behaviors.

For links in your application (for example, a media player), specify the URL for the media file in the standard format for CloudFront URLs. For more information, see the section called “Customize file URLs”.

Media quality-aware resiliency

Media quality-aware resiliency (MQAR) is an integrated capability between Amazon CloudFront and AWS Media Services. MQAR provides automated cross-Region origin selection based on Media Quality Confidence Score (MQCS). MQCS is synthesized by AWS Elemental MediaLive based on parameters that affect media quality experience perceived by viewers. You can configure CloudFront and AWS Media Services to deliver your live event streaming with high resiliency by using multiple options that you can specify in the CloudFront origin group failover criteria.

When you enable the MQAR feature for your distribution, you authorize CloudFront to automatically select the origin that is deemed to have the highest quality score.

The quality score represents perceived media streaming quality issues from your origins, such as black frames, frozen or dropped frames, or repeated frames. For example, if your AWS Elemental MediaPackage v2 origins are deployed in two different AWS Regions, and one reports a higher media quality score than the other, CloudFront will automatically switch to the origin that reports the higher score.

To achieve this, CloudFront does the following:

1. CloudFront forwards a `GET` request to the primary MediaPackage origin, and also initiates a `HEAD` request to the secondary MediaPackage origin at the same time. CloudFront receives the media quality score in the response headers from each origin.
2. Next, CloudFront tracks the score for each origin and uses this information to determine the origin with the higher score when a new request arrives.

The media quality score for your origins can change in real time. CloudFront determines this by consuming the MQCS changes, and switches between origins to make sure that viewers see the

higher media quality content. For more information, see Leveraging media quality scores with MediaPackage in the *AWS Elemental MediaPackage V2 User Guide*.

MQAR helps CloudFront determine, as early as possible, whether there's an issue that could potentially impact customers. For example, issues such as network connection, video processing, audio loss or drops, encoder speed problems can affect the media quality score for your viewers.

MQAR provides seamless switching between origins, so that you can deploy a resilient, cross-Region end-to-end media delivery workflow on AWS, and provide quality content for your viewers.

> **Note**
>
> Currently, this feature only supports MediaPackage v2 origins.

To enable this feature for your distribution, complete the following steps:

1. If you haven't already, create your MediaPackage v2 origins and enable this feature in your endpoint configuration. For a cross-region deployment, create a secondary channel in a different AWS Region with the same settings. For more information, see the following topics in the *AWS Elemental MediaPackage V2 User Guide*:
   - Create a channel and endpoint
   - Enable the media quality score
2. To use your MediaPackage v2 origins for CloudFront, create or update a CloudFront distribution. See Create a distribution and Update a distribution.
3. Create an origin group, and select your two origins as the primary and secondary. In your origin group, enable the **Media quality score** option. For more information, see Create an origin group.
4. In your cache behavior for your distribution, select the origin group that you created. We recommend that the cache behavior match the channel path pattern.

If CloudFront determines that both MediaPackage v2 origins have the same score, then it forwards the request to the primary origin as listed in the origin group. If the initially selected origin responds with an error code matching the failover criteria you specified in your origin group, then CloudFront retries the request to the alternative origin in your origin group regardless of its media quality score.

Notes

- CloudFront tracks the quality score for each cache behavior that utilizes an origin group enabled for media quality score. If the same origin group is used for multiple channels that emit a media quality score, create a separate cache behavior for each channel's path pattern to avoid mixing their scores. For more information about origin group quotas, see General quotas on distributions.
- Currently, MQAR isn't available when you use a Lambda@Edge function in origin-facing triggers (origin request and origin response) that is associated with your distribution's cache behavior. For more information, see Cache behavior settings.
- If you enabled the MQAR feature and origin access control (OAC), add the `mediapackagev2:GetHeadObject` action to the IAM policy. MQAR requires this permission to send HEAD requests to the MediaPackage v2 origin. For more information about OAC, see Restrict access to an AWS Elemental MediaPackage v2 origin.

MQAR log fields

CloudFront provides the following fields in real-time access logs to reflect the quality score and the selected origin. You can enable these fields in your CloudFront real-time access logs:

- `r-host`
- `sr-reason`
- `x-edge-mqcs`

For more information, see Fields 65-67.

Customize at the edge with functions

With Amazon CloudFront, you can write your own code to customize how your CloudFront distributions process HTTP requests and responses. The code runs close to your viewers (users) to minimize latency, and you don't have to manage servers or other infrastructure. You can write code to manipulate the requests and responses that flow through CloudFront, perform basic authentication and authorization, generate HTTP responses at the edge, and more.

The code that you write and attach to your CloudFront distribution is called an *edge function*. CloudFront provides two ways to write and manage edge functions:

CloudFront Functions

You can write lightweight functions in JavaScript for high-scale, latency-sensitive CDN customizations. The CloudFront Functions runtime environment offers submillisecond startup times, scales immediately to handle millions of requests per second, and is highly secure. CloudFront Functions is a native feature of CloudFront, which means you can build, test, and deploy your code entirely within CloudFront.

Lambda@Edge

Lambda@Edge is an extension of AWS Lambda that offers powerful and flexible computing for complex functions and full application logic closer to your viewers, and is highly secure. Lambda@Edge functions run in a Node.js or Python runtime environment. You publish them to a single AWS Region, but when you associate the function with a CloudFront distribution, Lambda@Edge automatically replicates your code around the world.

CloudFront Connection Functions

You can write lightweight JavaScript functions that run during the mTLS connection handshake to validate client certificates, implement custom authentication logic, and handle certificate revocation. Like CloudFront Functions, Connection Functions are a native feature of CloudFront that you can build, test, and deploy entirely within CloudFront. For more information, see Customize with CloudFront Connection Functions.

If you run AWS WAF on CloudFront, you can use AWS WAF inserted headers for both CloudFront Functions and Lambda@Edge. This works for viewer and origin requests and responses.

Topics

- Differences between CloudFront Functions and Lambda@Edge
- Customize at the edge with CloudFront Functions
- Customize with CloudFront Connection Functions
- Customize at the edge with Lambda@Edge
- Restrictions on edge functions

Differences between CloudFront Functions and Lambda@Edge

CloudFront Functions and Lambda@Edge both provide a way to run code in response to CloudFront events.

CloudFront Functions is ideal for lightweight, short-running functions for the following use cases:

- **Cache key normalization** – Transform HTTP request attributes (headers, query strings, cookies, and even the URL path) to create an optimal cache key, which can improve your cache hit ratio.
- **Header manipulation** – Insert, modify, or delete HTTP headers in the request or response. For example, you can add a `True-Client-IP` header to every request.
- **URL redirects or rewrites** – Redirect viewers to other pages based on information in the request, or rewrite all requests from one path to another.
- **Request authorization** – Validate hashed authorization tokens, such as JSON web tokens (JWT), by inspecting authorization headers or other request metadata.

To get started with CloudFront Functions, see Customize at the edge with CloudFront Functions.

Lambda@Edge is ideal for the following use cases:

- Functions that take several milliseconds or more to complete
- Functions that require adjustable CPU or memory
- Functions that depend on third-party libraries (including the AWS SDK, for integration with other AWS services)
- Functions that need more than one network call per invocation
- Functions that require file system access or access to the body of HTTP requests

To get started with Lambda@Edge, see Customize at the edge with Lambda@Edge.

To help you choose the option for your use case, use the following table to understand the differences between CloudFront Functions and Lambda@Edge. For information about differences that apply for origin modification helper methods, see Choose between CloudFront Functions and Lambda@Edge.

	CloudFront Functions	Lambda@Edge
Programming languages	JavaScript (ECMAScript 5.1 compliant)	Node.js and Python
Event sources	• Viewer request • Viewer response	• Viewer request • Viewer response • Origin request • Origin response
Supports Amazon CloudFront KeyValueStore	Yes CloudFront KeyValueStore only supports JavaScript runtime 2.0	No
Scale	Up to millions of requests per second	Up to 10,000 requests per second per Region
Function duration	Submillisecond	Up to 30 seconds (viewer request and viewer response) Up to 30 seconds (origin request and origin response)
Maximum function memory size	2 MB	128 MB (viewer request and viewer response) 10,240 MB (10 GB) (origin request and origin response)

	CloudFront Functions	Lambda@Edge
		For more informati on, see Quotas on Lambda@Edge.
Maximum size of the function code and included libraries	10 KB	50 MB (viewer request and viewer response) 50 MB (origin request and origin response)
Network access	Yes, one call per invocation to an origin configured on the same distribution	Yes, any destination
File system access	No	Yes
Access to the request body	No	Yes
Access to geolocation and device data	Yes	No (viewer request and viewer response) Yes (origin request and origin response)
Can build and test entirely within CloudFront	Yes	No
Function logging and metrics	Yes	Yes

Customize at the edge with CloudFront Functions

With CloudFront Functions, you can write lightweight functions in JavaScript for high-scale, latency-sensitive CDN customizations. Your functions can manipulate the requests and responses that flow through CloudFront, perform basic authentication and authorization, generate HTTP responses at the edge, and more. The CloudFront Functions runtime environment offers submillisecond startup times, scales immediately to handle millions of requests per second, and is

highly secure. CloudFront Functions is a native feature of CloudFront, which means you can build, test, and deploy your code entirely within CloudFront.

When you associate a CloudFront function with a CloudFront distribution, CloudFront intercepts requests and responses at CloudFront edge locations and passes them to your function. You can invoke CloudFront Functions when the following events occur:

- When CloudFront receives a request from a viewer (viewer request)
- Before CloudFront returns the response to the viewer (viewer response)
- During TLS connection establishment (connection request) - currently available for mutual TLS (mTLS) connections

For more information about CloudFront Functions, see the following topics. For a tutorial that stores data in a key value store, see Tutorial: Create a CloudFront function that includes key values. After you write, test, and publish a function, you associate it with a cache behavior in your distribution – see Associate functions with distributions.

Topics

- Tutorial: Create a simple function with CloudFront Functions
- Tutorial: Create a CloudFront function that includes key values
- Write function code
- Create functions
- Test functions
- Update functions
- Publish functions
- Associate functions with distributions
- Amazon CloudFront KeyValueStore

Tutorial: Create a simple function with CloudFront Functions

This tutorial shows you how to get started with CloudFront Functions. You can create a simple function that redirects the viewer to a different URL, and that also returns a custom response header.

Contents

- Prerequisites
- Create the function
- Verify the function

Prerequisites

To use CloudFront Functions, you need a CloudFront distribution. If you don't have one, see Get started with a CloudFront standard distribution.

Create the function

You can use the CloudFront console to create a simple function that redirects the viewer to a different URL, and also returns a custom response header.

To create a CloudFront function

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Functions**, and then choose **Create function**.
3. On the **Create function** page, for **Name**, enter a function name such as *MyFunctionName*.
4. (Optional) For **Description**, enter a description for the function such as **Simple test function**.
5. For **Runtime**, keep the default selected JavaScript version.
6. Choose **Create function**.
7. Copy the following function code. This function code redirects the viewer to a different URL, and also returns a custom response header.

```
function handler(event) {
    // NOTE: This example function is for a viewer request event trigger.
    // Choose viewer request for event trigger when you associate this function
 with a distribution.
    var response = {
        statusCode: 302,
        statusDescription: 'Found',
        headers: {
            'cloudfront-functions': { value: 'generated-by-CloudFront-Functions' },
            'location': { value: 'https://aws.amazon.com/cloudfront/' }
        }
```

```
    };
    return response;
}
```

8. For **Function code**, paste the code into the code editor to replace the default code.
9. Choose **Save changes**.
10. (Optional) You can test the function before you publish it. This tutorial doesn't describe how to test a function. For more information, see Test functions.
11. Choose the **Publish** tab and then choose **Publish function**. You *must* publish the function before you can associate it with your CloudFront distribution.
12. Next, you can associate the function with a distribution or cache behavior. On the *MyFunctionName* page, choose the **Publish** tab.

> **Warning**
>
> In the following steps, choose a distribution or a cache behavior that's used for testing. Don't associate this test function with a distribution or cache behavior that's used in production.

13. Choose **Add association**.
14. On the **Associate** dialog box, choose a distribution and/or a cache behavior. For **Event type**, keep the default value.
15. Choose **Add association**.

    The **Associated distributions** table shows the associated distribution.
16. Wait a few minutes for the associated distribution to finish deploying. To check the distribution's status, select the distribution in the **Associated distributions** table and then choose **View distribution**.

    When the distribution's status is **Deployed**, you're ready to verify that the function works.

Verify the function

After you deploy the function, you can verify that it's working for your distribution.

To verify the function

1. In your web browser, navigate to your distribution's domain name (for example, `https://d111111abcdef8.cloudfront.net`).

   The function returns a redirect to the browser, so the browser automatically goes to `https://aws.amazon.com/cloudfront/`.

2. In a command line window, you can use a tool like **curl** to send a request to your distribution's domain name.

   ```
   curl -v https://d111111abcdef8.cloudfront.net/
   ```

   In the response, you see the redirect response (`302 Found`) and the custom response headers that the function added. Your response might look like the following example.

   Example

   ```
   curl -v https://d111111abcdef8.cloudfront.net/
   > GET / HTTP/1.1
   > Host: d111111abcdef8.cloudfront.net
   > User-Agent: curl/7.64.1
   > Accept: */*
   >
   < HTTP/1.1 302 Found
   < Server: CloudFront
   < Date: Tue, 16 Mar 2021 18:50:48 GMT
   < Content-Length: 0
   < Connection: keep-alive
   < Location: https://aws.amazon.com/cloudfront/
   < Cloudfront-Functions: generated-by-CloudFront-Functions
   < X-Cache: FunctionGeneratedResponse from cloudfront
   < Via: 1.1 3035b31bddaf14eded329f8d22cf188c.cloudfront.net (CloudFront)
   < X-Amz-Cf-Pop: PHX50-C2
   < X-Amz-Cf-Id: ULZdIz6j43uGBlXyob_JctF9x7CCbwpNniiMlmNbmwzH1YWP9FsEHg==
   ```

Tutorial: Create a CloudFront function that includes key values

This tutorial shows you how to include key values with CloudFront function. Key values are part of a key-value pair. You include the name (from the key-value pair) in the function code. When the function runs, CloudFront replaces the name with the value.

Key-value pairs are variables that are stored in a key value store. When you use a key in your function (instead of hard-coded values), your function is more flexible. You can change the value of the key without having to deploy code changes. Key-value pairs can also reduce the size of your function. For more information, see ???.

Contents

- Prerequisites
- Create the key value store
- Add key-value pairs to the key value store
- Associate the key value store with the function
- Test and publish the function code

Prerequisites

If you're new to CloudFront Functions functions and the key value store, we recommend that you follow the tutorial in the section called “Tutorial: Create a simple CloudFront function”.

After you complete that tutorial, you can follow this tutorial to extend the function that you created. For this tutorial, we recommend that you create the key value store first.

Create the key value store

First, create the key value store to use for your function.

To create the key value store

1. Plan the key-value pairs you want to include in the function. Make a note of key names. The key-value pairs that you want to use in a function must be in a single key value store.
2. Decide about the order of work. There are two ways to proceed:

    - Create a key value store, and add key-value pairs to the store. Then create (or modify) the function and incorporate the key names.

- Or, create (or modify) the function and incorporate the key names you want to use. Then create a key value store, and add the key-value pairs.

3. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
4. In the navigation pane, choose **Functions**, and then choose the **KeyValueStores** tab.
5. Choose **Create KeyValueStore** and enter the following fields:

   - Enter a name and (optional) description for the store.
   - Leave **S3 URI** blank. In this tutorial you will enter the key-value pairs manually.

6. Choose **Create**. The details page for the new key value store appears. This page includes a **Key value pairs** section that is currently empty.

Add key-value pairs to the key value store

Next, manually add a list of key-value pairs to the key value store that you previously created.

To add key-value pairs to the key value store

1. In the **Key value pairs** section, choose **Add key value pairs**.
2. Choose **Add pair** and then enter a key and value. Choose the check mark to confirm your changes and repeat this step to add more.
3. When you're finished, choose **Save changes** to save the key-value pairs in the key value store. On the confirmation dialog, choose **Done**.

You now have a key value store that contains a group of key-value pairs.

Associate the key value store with the function

You have now created the key value store. And you have created or modified a function that includes the key names from the key value store. You can now associate the key value store and the function. You create that association from within the function.

To associate the key value store with the function

1. In the navigation pane, choose **Functions**. The **Functions** tab appears on top, by default.

2. Choose the function name and in the **Associated KeyValueStore** section, choose **Associate Existing KeyValueStore**.
3. Select the key value store and choose **Associate KeyValueStore**.

> **Note**
>
> You can associate only one key value store with each function.

Test and publish the function code

After you associate the key value store with your function, you can test and publish the function code. You should always test the function code every time you modify it, including when you do the following:

- Associate a key value store with the function.
- Modify the function and its key value store to include a new key-value pair.
- Change the value of a key-value pair.

To test and publish the function code

1. For information about how to test a function, see the section called “Test functions”. Make sure that you choose to test the function in the `DEVELOPMENT` stage.
2. Publish the function when you're ready to use the function (with the new or revised key value pairs) in a `LIVE` environment.

   When you publish, CloudFront copies the version of the function from the `DEVELOPMENT` stage over to the live stage. The function has the new code and is associated with the key value store. (There is no need to perform the association again, in the live stage.)

   For information about how to publish the function, see the section called “Publish functions”.

Write function code

You can use CloudFront Functions to write lightweight functions in JavaScript for high-scale, latency-sensitive CDN customizations. Your function code can manipulate the requests and

responses that flow through CloudFront, perform basic authentication and authorization, generate HTTP responses at the edge, and more.

To help you write function code for CloudFront Functions, see the following topics. For code examples, see CloudFront Functions examples for CloudFront and the amazon-cloudfront-functions repository on GitHub.

To use CBOR Web Tokens (CWT) for token-based authentication and authorization at CloudFront edge locations, see CWT support for CloudFront Functions.

Topics

- Determine function purpose
- CloudFront Functions event structure
- JavaScript runtime features for CloudFront Functions
- Helper methods for key value stores
- Helper methods for origin modification
- Helper methods for CloudFront SaaS Manager properties
- Use async and await
- CWT support for CloudFront Functions
- General helper methods
- Helper methods for mutual TLS
- Helper methods for network calls

Determine function purpose

Before you write your function code, determine the purpose of your function. Most functions in CloudFront Functions have one of the following purposes.

Topics

- Modify the HTTP request in a viewer request event type
- Generate an HTTP response in a viewer request event type
- Modify the HTTP response in a viewer response event type
- Validate mTLS connections in a connection request event type

- Related information

Regardless of your function's purpose, the `handler` is the entry point for any function. It takes a single argument called `event`, which is passed to the function by CloudFront. The `event` is a JSON object that contains a representation of the HTTP request (and the response, if your function modifies the HTTP response).

Modify the HTTP request in a viewer request event type

Your function can modify the HTTP request that CloudFront receives from the viewer (client), and return the modified request to CloudFront for continued processing. For example, your function code might normalize the cache key or modify request headers.

After you create and publish a function that modifies the HTTP request, make sure to add an association for the *viewer request* event type. For more information, see Create the function. This makes the function run each time that CloudFront receives a request from a viewer, before checking to see whether the requested object is in the CloudFront cache.

Example

The following pseudocode shows the structure of a function that modifies the HTTP request.

```
function handler(event) {
    var request = event.request;

    // Modify the request object here.

    return request;
}
```

The function returns the modified `request` object to CloudFront. CloudFront continues processing the returned request by checking the CloudFront cache for a cache hit, and sending the request to the origin if necessary.

Generate an HTTP response in a viewer request event type

Your function can generate an HTTP response at the edge and return it directly to the viewer (client) without checking for a cached response or any further processing by CloudFront. For example, your function code might redirect the request to a new URL, or check for authorization and return a `401` or `403` response to unauthorized requests.

When you create a function that generates an HTTP response, make sure to choose the *viewer request* event type. This means that the function runs each time CloudFront receives a request from a viewer, before CloudFront does any further processing of the request.

Example

The following pseudocode shows the structure of a function that generates an HTTP response.

```
function handler(event) {
    var request = event.request;

    var response = ...; // Create the response object here,
                        // using the request properties if needed.

    return response;
}
```

The function returns a `response` object to CloudFront, which CloudFront immediately returns to the viewer without checking the CloudFront cache or sending a request to the origin.

Modify the HTTP response in a viewer response event type

Your function can modify the HTTP response before CloudFront sends it to the viewer (client), regardless of whether the response comes from the CloudFront cache or the origin. For example, your function code might add or modify response headers, status codes, and body content.

When you create a function that modifies the HTTP response, make sure to choose the *viewer response* event type. This means that the function runs before CloudFront returns a response to the viewer, regardless of whether the response comes from the CloudFront cache or the origin.

Example

The following pseudocode shows the structure of a function that modifies the HTTP response.

```
function handler(event) {
    var request = event.request;
    var response = event.response;

    // Modify the response object here,
    // using the request properties if needed.
```

```
    return response;
}
```

The function returns the modified `response` object to CloudFront, which CloudFront immediately returns to the viewer.

Validate mTLS connections in a connection request event type

Connection functions are a type of CloudFront Functions that run during TLS connections to provide custom validation and authentication logic. Connection functions are currently available for mutual TLS (mTLS) connections, where you can validate client certificates and implement custom authentication logic beyond standard certificate validation. Connection functions run during the TLS handshake process and can allow or deny connections based on certificate properties, client IP addresses, or other criteria.

After you create and publish a connection function, make sure to add an association for the *connection request* event type with an mTLS-enabled distribution. This makes the function run each time a client attempts to establish an mTLS connection with CloudFront.

Example

The following pseudocode shows the structure of a connection function:

```
function connectionHandler(connection) {
    // Validate certificate and connection properties here.

    if (/* validation passes */) {
        connection.allow();
    } else {
        connection.deny();
    }
}
```

The function uses helper methods to determine whether to allow or deny the connection. Unlike viewer request and viewer response functions, connection functions cannot modify HTTP requests or responses.

Related information

For more information about working with CloudFront Functions, see the following topics:

- Event structure
- JavaScript runtime features
- CloudFront Functions examples
- Restrictions on edge functions

CloudFront Functions event structure

CloudFront Functions passes an `event` object to your function code as input when it runs the function. When you test a function, you create the `event` object and pass it to your function. When you create an `event` object for testing a function, you can omit the `distributionDomainName`, `distributionId`, and `requestId` fields in the `context` object. Make sure that the names of headers are lowercase, which is always the case in the `event` object that CloudFront Functions passes to your function in production.

The following shows an overview of the structure of this event object.

```
{
    "version": "1.0",
    "context": {
        <context object>
    },
    "viewer": {
        <viewer object>
    },
    "request": {
        <request object>
    },
    "response": {
        <response object>
    }
}
```

For more information, see the following topics:

Topics

- Version field
- Context object
- Connection event structure

- Viewer object
- Request object
- Response object
- Status code and body
- Structure for a query string, header, or cookie
- Example response object
- Example event object

Version field

The `version` field contains a string that specifies the version of the CloudFront Functions event object. The current version is `1.0`.

Context object

The `context` object contains contextual information about the event. It includes the following fields:

`distributionDomainName`

The CloudFront domain name (for example, d111111abcdef8.cloudfront.net) of the standard distribution that's associated with the event.

The `distributionDomainName` field only appears when your function is invoked for standard distributions.

`endpoint`

The CloudFront domain name (for example, d111111abcdef8.cloudfront.net) of the connection group that's associated with the event.

The `endpoint` field only appears when your function is invoked for multi-tenant distributions.

`distributionId`

The ID of the distribution (for example, EDFDVBD6EXAMPLE) that's associated with the event.

`eventType`

The event type, either `viewer-request` or `viewer-response`.

`requestId`

A string that uniquely identifies a CloudFront request (and its associated response).

Connection event structure

Connection functions receive a different event structure than viewer functions. For detailed information about the connection event structure and response format, see Associate a CloudFront Connection Function.

Viewer object

The `viewer` object contains an `ip` field whose value is the IP address of the viewer (client) that sent the request. If the viewer request came through an HTTP proxy or a load balancer, the value is the IP address of the proxy or load balancer.

Request object

The `request` object contains a representation of a viewer-to-CloudFront HTTP request. In the `event` object that's passed to your function, the `request` object represents the actual request that CloudFront received from the viewer.

If your function code returns a `request` object to CloudFront, it must use this same structure.

The `request` object contains the following fields:

`method`

The HTTP method of the request. If your function code returns a `request`, it can't modify this field. This is the only read-only field in the `request` object.

`uri`

The relative path of the requested object.

> **Note**
>
> If your function modifies the `uri` value, the following applies:
>
> - The new `uri` value must begin with a forward slash (`/`).
> - When a function changes the `uri` value, it changes the object that the viewer is requesting.

- When a function changes the `uri` value, it *doesn't* change the cache behavior for the request or the origin that an origin request is sent to.

`querystring`

An object that represents the query string in the request. If the request doesn't include a query string, the `request` object still includes an empty `querystring` object.

The `querystring` object contains one field for each query string parameter in the request.

`headers`

An object that represents the HTTP headers in the request. If the request contains any `Cookie` headers, those headers are not part of the `headers` object. Cookies are represented separately in the `cookies` object.

The `headers` object contains one field for each header in the request. Header names are converted to ASCII-lowercase in the event object, and header names must be ASCII-lowercase when they're added by your function code. When CloudFront Functions converts the event object back into an HTTP request, the first letter of each word in header names is capitalized, if it's an ASCII-letter. CloudFront Functions doesn't apply any changes to non-ASCII symbols in header names. For example, `TÈst-header` will become `tÈst-header` inside the function. The non-ASCII symbol È is unchanged.

Words are separated by a hyphen (-). For example, if your function code adds a header named `example-header-name`, CloudFront converts this to `Example-Header-Name` in the HTTP request.

`cookies`

An object that represents the cookies in the request (`Cookie` headers).

The `cookies` object contains one field for each cookie in the request.

For more information about the structure of query strings, headers, and cookies, see Structure for a query string, header, or cookie.

For an example `event` object, see Example event object.

Response object

The `response` object contains a representation of a CloudFront-to-viewer HTTP response. In the `event` object that's passed to your function, the `response` object represents CloudFront actual response to a viewer request.

If your function code returns a `response` object, it must use this same structure.

The `response` object contains the following fields:

`statusCode`

The HTTP status code of the response. This value is an integer, not a string.

Your function can generate or modify the `statusCode`.

`statusDescription`

The HTTP status description of the response. If your function code generates a response, this field is optional.

`headers`

An object that represents the HTTP headers in the response. If the response contains any `Set-Cookie` headers, those headers are not part of the `headers` object. Cookies are represented separately in the `cookies` object.

The `headers` object contains one field for each header in the response. Header names are converted to lowercase in the event object, and header names must be lowercase when they're added by your function code. When CloudFront Functions converts the event object back into an HTTP response, the first letter of each word in header names is capitalized. Words are separated by a hyphen (`-`). For example, if your function code adds a header named `example-header-name`, CloudFront converts this to `Example-Header-Name` in the HTTP response.

`cookies`

An object that represents the cookies in the response (`Set-Cookie` headers).

The `cookies` object contains one field for each cookie in the response.

`body`

Adding the `body` field is optional, and it will not be present in the `response` object unless you specify it in your function. Your function does not have access to the original body returned

by the CloudFront cache or origin. If you don't specify the body field in your viewer response function, the original body returned by the CloudFront cache or origin is returned to viewer.

If you want CloudFront to return a custom body to the viewer, specify the body content in the `data` field, and the body encoding in the `encoding` field. You can specify the encoding as plain text (`"encoding": "text"`) or as Base64-encoded content (`"encoding": "base64"`).

As a shortcut, you can also specify the body content directly in the body field (`"body": "<specify the body content here>"`). When you do this, omit the `data` and `encoding` fields. CloudFront treats the body as plain text in this case.

`encoding`

The encoding for the body content (`data` field). The only valid encodings are `text` and `base64`.

If you specify `encoding` as `base64` but the body is not valid base64, CloudFront returns an error.

`data`

The body content.

For more information about modified status codes and body content, see Status code and body.

For more information about the structure of headers and cookies, see Structure for a query string, header, or cookie.

For an example `response` object, see Example response object.

Status code and body

With CloudFront Functions, you can update the viewer response status code, replace the entire response body with a new one, or remove the response body. Some common scenarios for updating the viewer response after evaluating aspects of the response from the CloudFront cache or origin include the following:

- Changing the status to set an HTTP 200 status code and creating static body content to return to the viewer.
- Changing the status to set an HTTP 301 or 302 status code to redirect the user to another website.
- Deciding whether to serve or drop the body of the viewer response.

> **Note**
>
> If the origin returns an HTTP error of 400 and above, the CloudFront Function will not run. For more information see Restrictions on all edge functions.

When you're working with the HTTP response, CloudFront Functions does not have access to the response body. You can replace the body content by setting it to the desired value, or you can remove the body by setting the value to be empty. If you don't update the body field in your function, the original body returned by the CloudFront cache or origin is returned back to viewer.

> **Tip**
>
> When using CloudFront Functions to replace a body, be sure to align the corresponding headers, such as `content-encoding`, `content-type`, or `content-length`, to the new body content.
> For example, if the CloudFront origin or cache returns `content-encoding: gzip` but the viewer response function sets a body that's plain text, the function also needs to change the `content-encoding` and `content-type` headers accordingly.

If your CloudFront Function is configured to return an HTTP error of 400 or above, your viewer will not see a custom error page that you have specified for the same status code.

Structure for a query string, header, or cookie

Query strings, headers, and cookies share the same structure. Query strings can appear in requests. Headers appear in requests and responses. Cookies appear in requests and responses.

Each query string, header, or cookie is a unique field within the parent `querystring`, `headers`, or `cookies` object. The field name is the name of the query string, header, or cookie. Each field contains a `value` property with the value of the query string, header, or cookie.

Contents

- Query strings values or query string objects
- Special considerations for headers
- Duplicate query strings, headers, and cookies (multiValue array)
- Cookie attributes

Query strings values or query string objects

A function can return a query string value in addition to a query string object. The query string value can be used to arrange the query string parameters in any custom order.

Example

To modify a query string in your function code, use code like the following.

```
var request = event.request;
request.querystring =
 'ID=42&Exp=1619740800&TTL=1440&NoValue=&querymv=val1&querymv=val2,val3';
```

Special considerations for headers

For headers only, the header names are converted to lowercase in the event object, and header names must be lowercase when they're added by your function code. When CloudFront Functions converts the event object back into an HTTP request or response, the first letter of each word in header names is capitalized. Words are separated by a hyphen (-). For example, if your function code adds a header named `example-header-name`, CloudFront converts this to `Example-Header-Name` in the HTTP request or response.

Example

Consider the following `Host` header in an HTTP request.

```
Host: video.example.com
```

This header is represented as follows in the `request` object:

```
"headers": {
    "host": {
        "value": "video.example.com"
    }
}
```

To access the `Host` header in your function code, use code like the following:

```
var request = event.request;
var host = request.headers.host.value;
```

To add or modify a header in your function code, use code like the following (this code adds a header named `X-Custom-Header` with the value `example value`):

```
var request = event.request;
request.headers['x-custom-header'] = {value: 'example value'};
```

Duplicate query strings, headers, and cookies (`multiValue` array)

An HTTP request or response can contain more than one query string, header, or cookie with the same name. In this case, the duplicate query strings, headers, or cookies are collapsed into one field in the `request` or `response` object, but this field contains an extra property named `multiValue`. The `multiValue` property contains an array with the values of each of the duplicate query strings, headers, or cookies.

Example

Consider an HTTP request with the following `Accept` headers.

```
Accept: application/json
Accept: application/xml
Accept: text/html
```

These headers are represented as follows in the `request` object.

```
"headers": {
    "accept": {
        "value": "application/json",
        "multiValue": [
            {
                "value": "application/json"
            },
            {
                "value": "application/xml"
            },
            {
                "value": "text/html"
            }
        ]
    }
}
```

> **Note**
>
> The first header value (in this case, `application/json`) is repeated in both the `value` and `multiValue` properties. This allows you to access *all* the values by looping through the `multiValue` array.

If your function code modifies a query string, header, or cookie that has a `multiValue` array, CloudFront Functions uses the following rules to apply the changes:

1. If the `multiValue` array exists and has any modification, then that modification is applied. The first element in the `value` property is ignored.
2. Otherwise, any modification to the `value` property is applied, and subsequent values (if they exist) remain unchanged.

The `multiValue` property is used only when the HTTP request or response contains duplicate query strings, headers, or cookies with the same name, as shown in the preceding example. However, if there are multiple values in a single query string, header, or cookie, the `multiValue` property is not used.

Example

Consider a request with one `Accept` header that contains three values.

```
Accept: application/json, application/xml, text/html
```

This header is represented as follows in the `request` object.

```
"headers": {
    "accept": {
        "value": "application/json, application/xml, text/html"
    }
}
```

Cookie attributes

In a `Set-Cookie` header in an HTTP response, the header contains the name–value pair for the cookie and optionally a set of attributes separated by semicolons.

Example

```
Set-Cookie: cookie1=val1; Secure; Path=/; Domain=example.com; Expires=Wed, 05 Apr 2021
 07:28:00 GMT
```

In the `response` object, these attributes are represented in the `attributes` property of the cookie field. For example, the preceding `Set-Cookie` header is represented as follows:

```
"cookie1": {
    "value": "val1",
    "attributes": "Secure; Path=/; Domain=example.com; Expires=Wed, 05 Apr 2021
 07:28:00 GMT"
}
```

Example response object

The following example shows a `response` object — the output of a viewer response function — in which the body has been replaced by a viewer response function.

```
{
  "response": {
    "statusCode": 200,
    "statusDescription": "OK",
    "headers": {
      "date": {
        "value": "Mon, 04 Apr 2021 18:57:56 GMT"
      },
      "server": {
        "value": "gunicorn/19.9.0"
      },
      "access-control-allow-origin": {
        "value": "*"
      },
      "access-control-allow-credentials": {
        "value": "true"
      },
      "content-type": {
        "value": "text/html"
      },
      "content-length": {
        "value": "86"
```

```
      }
    },
    "cookies": {
      "ID": {
        "value": "id1234",
        "attributes": "Expires=Wed, 05 Apr 2021 07:28:00 GMT"
      },
      "Cookie1": {
        "value": "val1",
        "attributes": "Secure; Path=/; Domain=example.com; Expires=Wed, 05 Apr 2021
 07:28:00 GMT",
        "multiValue": [
          {
            "value": "val1",
            "attributes": "Secure; Path=/; Domain=example.com; Expires=Wed, 05 Apr 2021
 07:28:00 GMT"
          },
          {
            "value": "val2",
            "attributes": "Path=/cat; Domain=example.com; Expires=Wed, 10 Jan 2021
 07:28:00 GMT"
          }
        ]
      }
    },

    // Adding the body field is optional and it will not be present in the response
 object
    // unless you specify it in your function.
    // Your function does not have access to the original body returned by the
 CloudFront
    // cache or origin.
    // If you don't specify the body field in your viewer response function, the
 original
    // body returned by the CloudFront cache or origin is returned to viewer.

     "body": {
      "encoding": "text",
      "data": "<!DOCTYPE html><html><body><p>Here is your custom content.</p></body></
html>"
    }
  }
}
```

Example event object

The following example shows a complete `event` object. This is an example invocation for a standard distribution, and not for a multi-tenant distribution. For multi-tenant distributions, the `endpoint` field is used instead of `distributionDomainName`. The value of `endpoint` is the CloudFront domain name (for example, d111111abcdef8.cloudfront.net) of the connection group that's associated with the event.

> **Note**
>
> The `event` object is the input to your function. Your function returns only the `request` or `response` object, not the complete `event` object.

```
{
    "version": "1.0",
    "context": {
        "distributionDomainName": "d111111abcdef8.cloudfront.net",
        "distributionId": "EDFDVBD6EXAMPLE",
        "eventType": "viewer-response",
        "requestId": "EXAMPLEntjQpEXAMPLE_SG5Z-EXAMPLEPmPfEXAMPLEu3EqEXAMPLE=="
    },
    "viewer": {"ip": "198.51.100.11"},
    "request": {
        "method": "GET",
        "uri": "/media/index.mpd",
        "querystring": {
            "ID": {"value": "42"},
            "Exp": {"value": "1619740800"},
            "TTL": {"value": "1440"},
            "NoValue": {"value": ""},
            "querymv": {
                "value": "val1",
                "multiValue": [
                    {"value": "val1"},
                    {"value": "val2,val3"}
                ]
            }
        },
        "headers": {
            "host": {"value": "video.example.com"},
```

```
            "user-agent": {"value": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0)
 Gecko/20100101 Firefox/83.0"},
            "accept": {
                "value": "application/json",
                "multiValue": [
                    {"value": "application/json"},
                    {"value": "application/xml"},
                    {"value": "text/html"}
                ]
            },
            "accept-language": {"value": "en-GB,en;q=0.5"},
            "accept-encoding": {"value": "gzip, deflate, br"},
            "origin": {"value": "https://website.example.com"},
            "referer": {"value": "https://website.example.com/videos/12345678?
action=play"},
            "cloudfront-viewer-country": {"value": "GB"}
        },
        "cookies": {
            "Cookie1": {"value": "value1"},
            "Cookie2": {"value": "value2"},
            "cookie_consent": {"value": "true"},
            "cookiemv": {
                "value": "value3",
                "multiValue": [
                    {"value": "value3"},
                    {"value": "value4"}
                ]
            }
        }
    },
    "response": {
        "statusCode": 200,
        "statusDescription": "OK",
        "headers": {
            "date": {"value": "Mon, 04 Apr 2021 18:57:56 GMT"},
            "server": {"value": "gunicorn/19.9.0"},
            "access-control-allow-origin": {"value": "*"},
            "access-control-allow-credentials": {"value": "true"},
            "content-type": {"value": "application/json"},
            "content-length": {"value": "701"}
        },
        "cookies": {
            "ID": {
                "value": "id1234",
```

```
                "attributes": "Expires=Wed, 05 Apr 2021 07:28:00 GMT"
            },
            "Cookie1": {
                "value": "val1",
                "attributes": "Secure; Path=/; Domain=example.com; Expires=Wed, 05 Apr
 2021 07:28:00 GMT",
                "multiValue": [
                    {
                        "value": "val1",
                        "attributes": "Secure; Path=/; Domain=example.com; Expires=Wed,
 05 Apr 2021 07:28:00 GMT"
                    },
                    {
                        "value": "val2",
                        "attributes": "Path=/cat; Domain=example.com; Expires=Wed, 10
 Jan 2021 07:28:00 GMT"
                    }
                ]
            }
        }
    }
}
```

JavaScript runtime features for CloudFront Functions

The CloudFront Functions JavaScript runtime environment is compliant with ECMAScript (ES) version 5.1 and also supports some features of ES versions 6 through 12.

For the most up-to-date features, we recommend that you use JavaScript runtime 2.0.

The JavaScript runtime 2.0 features has the following changes compared to 1.0:

- Buffer module methods are available
- The following non-standard string prototype methods are not available:
  - `String.prototype.bytesFrom()`
  - `String.prototype.fromBytes()`
  - `String.prototype.fromUTF8()`
  - `String.prototype.toBytes()`
  - `String.prototype.toUTF8()`
- The cryptographic module has the following changes:

- `hash.digest()` – Return type is changed to `Buffer` if no encoding is provided
- `hmac.digest()` – Return type is changed to `Buffer` if no encoding is provided

- For more information about additional new features, see JavaScript runtime 2.0 features for CloudFront Functions.

Topics

- JavaScript runtime 1.0 features for CloudFront Functions
- JavaScript runtime 2.0 features for CloudFront Functions

JavaScript runtime 1.0 features for CloudFront Functions

The CloudFront Functions JavaScript runtime environment is compliant with ECMAScript (ES) version 5.1 and also supports some features of ES versions 6 through 9. It also provides some nonstandard methods that are not part of the ES specifications.

The following topics list all the supported language features.

Topics

- Core features
- Primitive objects
- Built-in objects
- Error types
- Globals
- Built-in modules
- Restricted features

Core features

The following core features of ES are supported.

Types

All ES 5.1 types are supported. This includes Boolean values, numbers, strings, objects, arrays, functions, function constructors, and regular expressions.

Operators

All ES 5.1 operators are supported.

The ES 7 exponentiation operator (**) is supported.

Statements

> **Note**
>
> The `const` and `let` statements are not supported.

The following ES 5.1 statements are supported:

- `break`
- `catch`
- `continue`
- `do-while`
- `else`
- `finally`
- `for`
- `for-in`
- `if`
- `return`
- `switch`
- `throw`
- `try`
- `var`
- `while`
- Labeled statements

Literals

ES 6 template literals are supported: multiline strings, expression interpolation, and nesting templates.

Functions

All ES 5.1 function features are supported.

ES 6 arrow functions are supported, and ES 6 rest parameter syntax is supported.

Unicode

Source text and string literals can contain Unicode-encoded characters. Unicode code point escape sequences of six characters (for example, \uXXXX) are also supported.

Strict mode

Functions operate in strict mode by default, so you don't need to add a `use strict` statement in your function code. This cannot be changed.

Primitive objects

The following primitive objects of ES are supported.

Object

The following ES 5.1 methods on objects are supported:

- `create` (without properties list)
- `defineProperties`
- `defineProperty`
- `freeze`
- `getOwnPropertyDescriptor`
- `getOwnPropertyNames`
- `getPrototypeOf`
- `hasOwnProperty`
- `isExtensible`
- `isFrozen`
- `prototype.isPrototypeOf`
- `isSealed`
- `keys`

- preventExtensions
- prototype.propertyIsEnumerable
- seal
- prototype.toString
- prototype.valueOf

The following ES 6 methods on objects are supported:

- assign
- is
- prototype.setPrototypeOf

The following ES 8 methods on objects are supported:

- entries
- values

String

The following ES 5.1 methods on strings are supported:

- fromCharCode
- prototype.charAt
- prototype.concat
- prototype.indexOf
- prototype.lastIndexOf
- prototype.match
- prototype.replace
- prototype.search
- prototype.slice
- prototype.split
- prototype.substr
- prototype.substring
- prototype.toLowerCase
- prototype.trim

- `prototype.toUpperCase`

The following ES 6 methods on strings are supported:

- `fromCodePoint`
- `prototype.codePointAt`
- `prototype.endsWith`
- `prototype.includes`
- `prototype.repeat`
- `prototype.startsWith`

The following ES 8 methods on strings are supported:

- `prototype.padStart`
- `prototype.padEnd`

The following ES 9 methods on strings are supported:

- `prototype.trimStart`
- `prototype.trimEnd`

The following nonstandard methods on strings are supported:

- `prototype.bytesFrom(`*`array | string, encoding`*`)`

  Creates a byte string from an array of octets or an encoded string. The string encoding options are `hex`, `base64`, and `base64url`.

- `prototype.fromBytes(`*`start`*`[, `*`end`*`])`

  Creates a Unicode string from a byte string where each byte is replaced with the corresponding Unicode code point.

- `prototype.fromUTF8(`*`start`*`[, `*`end`*`])`

  Creates a Unicode string from a UTF-8 encoded byte string. If the encoding is incorrect, it returns `null`.

- `prototype.toBytes(start[, end])`

  Creates a byte string from a Unicode string. All characters must be in the [0,255] range. If not, it returns `null`.

- `prototype.toUTF8(start[, end])`

  Creates a UTF-8 encoded byte string from a Unicode string.

Number

All ES 5.1 methods on numbers are supported.

The following ES 6 methods on numbers are supported:

- `isFinite`
- `isInteger`
- `isNaN`
- `isSafeInteger`
- `parseFloat`
- `parseInt`
- `prototype.toExponential`
- `prototype.toFixed`
- `prototype.toPrecision`
- `EPSILON`
- `MAX_SAFE_INTEGER`
- `MAX_VALUE`
- `MIN_SAFE_INTEGER`
- `MIN_VALUE`
- `NEGATIVE_INFINITY`
- `NaN`
- `POSITIVE_INFINITY`

Built-in objects

The following built-in objects of ES are supported.

Math

All ES 5.1 math methods are supported.

> **Note**
>
> In the CloudFront Functions runtime environment, the `Math.random()` implementation uses OpenBSD `arc4random` seeded with the timestamp of when the function runs.

The following ES 6 math methods are supported:

- `acosh`
- `asinh`
- `atanh`
- `cbrt`
- `clz32`
- `cosh`
- `expm1`
- `fround`
- `hypot`
- `imul`
- `log10`
- `log1p`
- `log2`
- `sign`
- `sinh`
- `tanh`
- `trunc`
- `E`
- `LN10`
- `LN2`
- `LOG10E`
- `LOG2E`
- `PI`

- SQRT1_2
- SQRT2

Date

All ES 5.1 Date features are supported.

> **Note**
>
> For security reasons, Date always returns the same value—the function's start time—during the lifetime of a single function run. For more information, see Restricted features.

Function

The apply, bind, and call methods are supported.

Function constructors are not supported.

Regular expressions

All ES 5.1 regular expression features are supported. The regular expression language is Perl compatible. ES 9 named capture groups are supported.

JSON

All ES 5.1 JSON features are supported, including parse and stringify.

Array

The following ES 5.1 methods on arrays are supported:

- isArray
- prototype.concat
- prototype.every
- prototype.filter
- prototype.forEach
- prototype.indexOf
- prototype.join
- prototype.lastIndexOf

- prototype.map
- prototype.pop
- prototype.push
- prototype.reduce
- prototype.reduceRight
- prototype.reverse
- prototype.shift
- prototype.slice
- prototype.some
- prototype.sort
- prototype.splice
- prototype.unshift

The following ES 6 methods on arrays are supported:

- of
- prototype.copyWithin
- prototype.fill
- prototype.find
- prototype.findIndex

The following ES 7 methods on arrays are supported:

- prototype.includes

Typed arrays

The following ES 6 typed arrays are supported:

- Int8Array
- Uint8Array
- Uint8ClampedArray
- Int16Array
- Uint16Array
- Int32Array
- Uint32Array

- `Float32Array`
- `Float64Array`
- `prototype.copyWithin`
- `prototype.fill`
- `prototype.join`
- `prototype.set`
- `prototype.slice`
- `prototype.subarray`
- `prototype.toString`

ArrayBuffer

The following methods on `ArrayBuffer` are supported:

- `prototype.isView`
- `prototype.slice`

Promise

The following methods on promises are supported:

- `reject`
- `resolve`
- `prototype.catch`
- `prototype.finally`
- `prototype.then`

Crypto

The cryptographic module provides standard hashing and hash-based message authentication code (HMAC) helpers. You can load the module using `require('crypto')`. The module exposes the following methods that behave exactly as their Node.js counterparts:

- `createHash(`*`algorithm`*`)`
- `hash.update(`*`data`*`)`
- `hash.digest([`*`encoding`*`])`
- `createHmac(`*`algorithm, secret key`*`)`
- `hmac.update(`*`data`*`)`

- `hmac.digest([encoding])`

For more information, see Crypto (hash and HMAC) in the built-in modules section.

Console

This is a helper object for debugging. It only supports the `log()` method, to record log messages.

> **Note**
>
> CloudFront Functions doesn't support comma syntax, such as `console.log('a', 'b')`. Instead, use the `console.log('a' + ' ' + 'b')` format.

Error types

The following error objects are supported:

- `Error`
- `EvalError`
- `InternalError`
- `MemoryError`
- `RangeError`
- `ReferenceError`
- `SyntaxError`
- `TypeError`
- `URIError`

Globals

The `globalThis` object is supported.

The following ES 5.1 global functions are supported:

- `decodeURI`
- `decodeURIComponent`

- `encodeURI`
- `encodeURIComponent`
- `isFinite`
- `isNaN`
- `parseFloat`
- `parseInt`

The following global constants are supported:

- `NaN`
- `Infinity`
- `undefined`

Built-in modules

The following built-in modules are supported.

Modules

- Crypto (hash and HMAC)
- Query string

Crypto (hash and HMAC)

The cryptographic module (`crypto`) provides standard hashing and hash-based message authentication code (HMAC) helpers. You can load the module using `require('crypto')`. The module provides the following methods that behave exactly as their Node.js counterparts.

Hashing methods

`crypto.createHash(`*`algorithm`*`)`

Creates and returns a hash object that you can use to generate hash digests using the given algorithm: `md5`, `sha1`, or `sha256`.

`hash.update(`*`data`*`)`

Updates the hash content with the given `data`.

`hash.digest([encoding])`

Calculates the digest of all of the data passed using `hash.update()`. The encoding can be `hex`, `base64`, or `base64url`.

HMAC methods

`crypto.createHmac(algorithm, secret key)`

Creates and returns an HMAC object that uses the given `algorithm` and `secret key`. The algorithm can be `md5`, `sha1`, or `sha256`.

`hmac.update(data)`

Updates the HMAC content with the given `data`.

`hmac.digest([encoding])`

Calculates the digest of all of the data passed using `hmac.update()`. The encoding can be `hex`, `base64`, or `base64url`.

Query string

> **Note**
>
> The CloudFront Functions event object automatically parses URL query strings for you. That means that in most cases you don't need to use this module.

The query string module (`querystring`) provides methods for parsing and formatting URL query strings. You can load the module using `require('querystring')`. The module provides the following methods.

`querystring.escape(string)`

URL-encodes the given `string`, returning an escaped query string. The method is used by `querystring.stringify()` and should not be used directly.

`querystring.parse(string[, separator[, equal[, options]]])`

Parses a query string (`string`) and returns an object.

The `separator` parameter is a substring for delimiting key and value pairs in the query string. By default it is &.

The `equal` parameter is a substring for delimiting keys and values in the query string. By default it is =.

The `options` parameter is an object with the following keys:

`decodeURIComponent` *`function`*

A function to decode percent-encoded characters in the query string. By default it is `querystring.unescape()`.

`maxKeys` *`number`*

The maximum number of keys to parse. By default it is `1000`. Use a value of `0` to remove the limitations for counting keys.

By default, percent-encoded characters within the query string are assumed to use the UTF-8 encoding. Invalid UTF-8 sequences are replaced with the U+FFFD replacement character.

For example, for the following query string:

```
'name=value&abc=xyz&abc=123'
```

The return value of `querystring.parse()` is:

```
{
name: 'value',
abc: ['xyz', '123']
}
```

`querystring.decode()` is an alias for `querystring.parse()`.

`querystring.stringify(`*`object`*`[, `*`separator`*`[, `*`equal`*`[, `*`options`*`]]])`

Serializes an `object` and returns a query string.

The `separator` parameter is a substring for delimiting key and value pairs in the query string. By default it is &.

The `equal` parameter is a substring for delimiting keys and values in the query string. By default it is =.

The `options` parameter is an object with the following keys:

`encodeURIComponent` *`function`*

The function to use for converting URL-unsafe characters to percent-encoding in the query string. By default it is `querystring.escape()`.

By default, characters that require percent-encoding within the query string are encoded as UTF-8. To use a different encoding, specify the `encodeURIComponent` option.

For example, for the following code:

```
querystring.stringify({ name: 'value', abc: ['xyz', '123'], anotherName: '' });
```

The return value is:

```
'name=value&abc=xyz&abc=123&anotherName='
```

`querystring.encode()` is an alias for `querystring.stringify()`.

`querystring.unescape(`*`string`*`)`

Decodes URL percent-encoded characters in the given `string`, returning an unescaped query string. This method is used by `querystring.parse()` and should not be used directly.

Restricted features

The following JavaScript language features are either unsupported or restricted due to security concerns.

Dynamic code evaluation

Dynamic code evaluation is not supported. Both `eval()` and `Function` constructors throw an error if attempted. For example, `const sum = new Function('a', 'b', 'return a + b')` throws an error.

Timers

The `setTimeout()`, `setImmediate()`, and `clearTimeout()` functions are not supported. There is no provision to defer or yield within a function run. Your function must synchronously run to completion.

Date and timestamps

For security reasons, there is no access to high-resolution timers. All `Date` methods to query the current time always return the same value during the lifetime of a single function run. The returned timestamp is the time when the function started running. Consequently, you cannot measure elapsed time in your function.

File system access

There is no file system access. For example, there is no `fs` module for file system access like there is in Node.js.

Process access

There is no process access. For example, there is no `process` global object for processing information access like there is in Node.js.

Environment variables

There is no access to environment variables.

Instead, you can use CloudFront KeyValueStore to create a centralized datastore of key-value pairs for your CloudFront Functions. CloudFront KeyValueStore enables dynamic updates to your configuration data without needing to deploy code changes. You must use JavaScript runtime 2.0 to use CloudFront KeyValueStore. For more information, see Amazon CloudFront KeyValueStore.

Network access

There is no support for network calls. For example, XHR, HTTP(S), and socket are not supported.

JavaScript runtime 2.0 features for CloudFront Functions

The CloudFront Functions JavaScript runtime environment is compliant with ECMAScript (ES) version 5.1 and also supports some features of ES versions 6 through 12. It also provides some nonstandard methods that are not part of the ES specifications. The following topics list all supported features in this runtime.

Topics

- Core features
- Primitive objects

- Built-in objects
- Error types
- Globals
- Built-in modules
- Restricted features

Core features

The following core features of ES are supported.

Types

All ES 5.1 types are supported. This includes boolean values, numbers, strings, objects, arrays, functions, and regular expressions.

Operators

All ES 5.1 operators are supported.

The ES 7 exponentiation operator (`**`) is supported.

Statements

The following ES 5.1 statements are supported:

- `break`
- `catch`
- `continue`
- `do-while`
- `else`
- `finally`
- `for`
- `for-in`
- `if`
- `label`
- `return`
- `switch`

- `throw`
- `try`
- `var`
- `while`

The following ES 6 statements are supported:

- `const`
- `let`

The following ES 8 statements are supported:

- `async`
- `await`

> **Note**
>
> `async`, `await`, `const`, and `let` are supported in JavaScript runtime 2.0.
> `await` can be used inside `async` functions only. `async` arguments and closures are not supported.

Literals

ES 6 template literals are supported: multiline strings, expression interpolation, and nesting templates.

Functions

All ES 5.1 function features are supported.

ES 6 arrow functions are supported, and ES 6 rest parameter syntax is supported.

Unicode

Source text and string literals can contain Unicode-encoded characters. Unicode code point escape sequences of six characters (for example, \uXXXX) are also supported.

Strict mode

Functions operate in strict mode by default, so you don't need to add a `use strict` statement in your function code. This cannot be changed.

Primitive objects

The following primitive objects of ES are supported.

Object

The following ES 5.1 methods on objects are supported:

- `Object.create()` (without properties list)
- `Object.defineProperties()`
- `Object.defineProperty()`
- `Object.freeze()`
- `Object.getOwnPropertyDescriptor()`
- `Object.getOwnPropertyDescriptors()`
- `Object.getOwnPropertyNames()`
- `Object.getPrototypeOf()`
- `Object.isExtensible()`
- `Object.isFrozen()`
- `Object.isSealed()`
- `Object.keys()`
- `Object.preventExtensions()`
- `Object.seal()`

The following ES 6 methods on objects are supported:

- `Object.assign()`

The following ES 8 methods on objects are supported:

- `Object.entries()`
- `Object.values()`

The following ES 5.1 prototype methods on objects are supported:

- `Object.prototype.hasOwnProperty()`
- `Object.prototype.isPrototypeOf()`
- `Object.prototype.propertyIsEnumerable()`
- `Object.prototype.toString()`

- `Object.prototype.valueOf()`

The following ES 6 prototype methods on objects are supported:

- `Object.prototype.is()`
- `Object.prototype.setPrototypeOf()`

String

The following ES 5.1 methods on strings are supported:

- `String.fromCharCode()`

The following ES 6 methods on strings are supported:

- `String.fromCodePoint()`

The following ES 5.1 prototype methods on strings are supported:

- `String.prototype.charAt()`
- `String.prototype.concat()`
- `String.prototype.indexOf()`
- `String.prototype.lastIndexOf()`
- `String.prototype.match()`
- `String.prototype.replace()`
- `String.prototype.search()`
- `String.prototype.slice()`
- `String.prototype.split()`
- `String.prototype.substr()`
- `String.prototype.substring()`
- `String.prototype.toLowerCase()`
- `String.prototype.trim()`
- `String.prototype.toUpperCase()`

The following ES 6 prototype methods on strings are supported:

- `String.prototype.codePointAt()`
- `String.prototype.endsWith()`
- `String.prototype.includes()`

- `String.prototype.repeat()`
- `String.prototype.startsWith()`

The following ES 8 prototype methods on strings are supported:

- `String.prototype.padStart()`
- `String.prototype.padEnd()`

The following ES 9 prototype methods on strings are supported:

- `String.prototype.trimStart()`
- `String.prototype.trimEnd()`

The following ES 12 prototype methods on strings are supported:

- `String.prototype.replaceAll()`

> **Note**
>
> `String.prototype.replaceAll()` is new in JavaScript runtime 2.0.

Number

All ES 5 numbers are supported.

The following ES 6 properties on numbers are supported:

- `Number.EPSILON`
- `Number.MAX_SAFE_INTEGER`
- `Number.MIN_SAFE_INTEGER`
- `Number.MAX_VALUE`
- `Number.MIN_VALUE`
- `Number.NaN`
- `Number.NEGATIVE_INFINITY`
- `Number.POSITIVE_INFINITY`

The following ES 6 methods on numbers are supported:

- `Number.isFinite()`
- `Number.isInteger()`

- `Number.isNaN()`
- `Number.isSafeInteger()`
- `Number.parseInt()`
- `Number.parseFloat()`

The following ES 5.1 prototype methods on numbers are supported:

- `Number.prototype.toExponential()`
- `Number.prototype.toFixed()`
- `Number.prototype.toPrecision()`

ES 12 numeric separators are supported.

> **Note**
>
> ES 12 numeric separators are new in JavaScript runtime 2.0.

Built-in objects

The following built-in objects of ES are supported.

Math

All ES 5.1 math methods are supported.

> **Note**
>
> In the CloudFront Functions runtime environment, the `Math.random()` implementation uses OpenBSD `arc4random` seeded with the timestamp of when the function runs.

The following ES 6 math properties are supported:

- `Math.E`
- `Math.LN10`
- `Math.LN2`
- `Math.LOG10E`

- `Math.LOG2E`
- `Math.PI`
- `Math.SQRT1_2`
- `Math.SQRT2`

The following ES 6 math methods are supported:

- `Math.abs()`
- `Math.acos()`
- `Math.acosh()`
- `Math.asin()`
- `Math.asinh()`
- `Math.atan()`
- `Math.atan2()`
- `Math.atanh()`
- `Math.cbrt()`
- `Math.ceil()`
- `Math.clz32()`
- `Math.cos()`
- `Math.cosh()`
- `Math.exp()`
- `Math.expm1()`
- `Math.floor()`
- `Math.fround()`
- `Math.hypot()`
- `Math.imul()`
- `Math.log()`
- `Math.log1p()`
- `Math.log2()`
- `Math.log10()`
- `Math.max()`
- `Math.min()`

- `Math.pow()`
- `Math.random()`
- `Math.round()`
- `Math.sign()`
- `Math.sinh()`
- `Math.sin()`
- `Math.sqrt()`
- `Math.tan()`
- `Math.tanh()`
- `Math.trunc()`

Date

All ES 5.1 `Date` features are supported.

> **Note**
>
> For security reasons, `Date` always returns the same value—the function's start time—during the lifetime of a single function run. For more information, see Restricted features.

Function

The following ES 5.1 prototype methods are supported:

- `Function.prototype.apply()`
- `Function.prototype.bind()`
- `Function.prototype.call()`

Function constructors are not supported.

Regular expressions

All ES 5.1 regular expression features are supported. The regular expression language is Perl compatible.

The following ES 5.1 prototype accessor properties are supported:

- `RegExp.prototype.global`
- `RegExp.prototype.ignoreCase`
- `RegExp.prototype.multiline`
- `RegExp.prototype.source`
- `RegExp.prototype.sticky`
- `RegExp.prototype.flags`

> **Note**
>
> `RegExp.prototype.sticky` and `RegExp.prototype.flags` are new in JavaScript runtime 2.0.

The following ES 5.1 prototype methods are supported:

- `RegExp.prototype.exec()`
- `RegExp.prototype.test()`
- `RegExp.prototype.toString()`
- `RegExp.prototype[@@replace]()`
- `RegExp.prototype[@@split]()`

> **Note**
>
> `RegExp.prototype[@@split]()` is new in JavaScript runtime 2.0.

The following ES 5.1 instance properties are supported:

- `lastIndex`

ES 9 named capture groups are supported.

JSON

The following ES 5.1 methods are supported:

- `JSON.parse()`
- `JSON.stringify()`

Array

The following ES 5.1 methods on arrays are supported:

- `Array.isArray()`

The following ES 6 methods on arrays are supported:

- `Array.of()`

The following ES 5.1 prototype methods are supported:

- `Array.prototype.concat()`
- `Array.prototype.every()`
- `Array.prototype.filter()`
- `Array.prototype.forEach()`
- `Array.prototype.indexOf()`
- `Array.prototype.join()`
- `Array.prototype.lastIndexOf()`
- `Array.prototype.map()`
- `Array.prototype.pop()`
- `Array.prototype.push()`
- `Array.prototype.reduce()`
- `Array.prototype.reduceRight()`
- `Array.prototype.reverse()`
- `Array.prototype.shift()`
- `Array.prototype.slice()`
- `Array.prototype.some()`
- `Array.prototype.sort()`
- `Array.prototype.splice()`
- `Array.prototype.unshift()`

The following ES 6 prototype methods are supported

- `Array.prototype.copyWithin()`
- `Array.prototype.fill()`
- `Array.prototype.find()`
- `Array.prototype.findIndex()`

The following ES 7 prototype methods are supported:

- `Array.prototype.includes()`

Typed arrays

The following ES 6 typed array constructors are supported:

- `Float32Array`
- `Float64Array`
- `Int8Array`
- `Int16Array`
- `Int32Array`
- `Uint8Array`
- `Uint8ClampedArray`
- `Uint16Array`
- `Uint32Array`

The following ES 6 methods are supported:

- `TypedArray.from()`
- `TypedArray.of()`

> **Note**
>
> `TypedArray.from()` and `TypedArray.of()` are new in JavaScript runtime 2.0.

The following ES 6 prototype methods are supported:

- `TypedArray.prototype.copyWithin()`
- `TypedArray.prototype.every()`
- `TypedArray.prototype.fill()`
- `TypedArray.prototype.filter()`
- `TypedArray.prototype.find()`
- `TypedArray.prototype.findIndex()`
- `TypedArray.prototype.forEach()`
- `TypedArray.prototype.includes()`
- `TypedArray.prototype.indexOf()`

- `TypedArray.prototype.join()`
- `TypedArray.prototype.lastIndexOf()`
- `TypedArray.prototype.map()`
- `TypedArray.prototype.reduce()`
- `TypedArray.prototype.reduceRight()`
- `TypedArray.prototype.reverse()`
- `TypedArray.prototype.some()`
- `TypedArray.prototype.set()`
- `TypedArray.prototype.slice()`
- `TypedArray.prototype.sort()`
- `TypedArray.prototype.subarray()`
- `TypedArray.prototype.toString()`

> **Note**
>
> `TypedArray.prototype.every()`, `TypedArray.prototype.fill()`, `TypedArray.prototype.filter()`, `TypedArray.prototype.find()`, `TypedArray.prototype.findIndex()`, `TypedArray.prototype.forEach()`, `TypedArray.prototype.includes()`, `TypedArray.prototype.indexOf()`, `TypedArray.prototype.join()`, `TypedArray.prototype.lastIndexOf()`,`TypedArray.prototype.map()`, `TypedArray.prototype.reduce()`, `TypedArray.prototype.reduceRight()`, `TypedArray.prototype.reverse()`, and `TypedArray.prototype.some()` are new in JavaScript runtime 2.0.

ArrayBuffer

The following ES 6 methods on ArrayBuffer are supported:

- `isView()`

The following ES 6 prototype methods on ArrayBuffer are supported:

- `ArrayBuffer.prototype.slice()`

Promise

The following ES 6 methods on promises are supported:

- `Promise.all()`
- `Promise.allSettled()`
- `Promise.any()`
- `Promise.reject()`
- `Promise.resolve()`
- `Promise.race()`

> **Note**
>
> `Promise.all()`, `Promise.allSettled()`, `Promise.any()`, and `Promise.race()` are new in JavaScript runtime 2.0.

The following ES 6 prototype methods on promises are supported:

- `Promise.prototype.catch()`
- `Promise.prototype.finally()`
- `Promise.prototype.then()`

DataView

The following ES 6 prototype methods are supported:

- `DataView.prototype.getFloat32()`
- `DataView.prototype.getFloat64()`
- `DataView.prototype.getInt16()`
- `DataView.prototype.getInt32()`
- `DataView.prototype.getInt8()`
- `DataView.prototype.getUint16()`
- `DataView.prototype.getUint32()`
- `DataView.prototype.getUint8()`
- `DataView.prototype.setFloat32()`
- `DataView.prototype.setFloat64()`
- `DataView.prototype.setInt16()`
- `DataView.prototype.setInt32()`
- `DataView.prototype.setInt8()`

- `DataView.prototype.setUint16()`
- `DataView.prototype.setUint32()`
- `DataView.prototype.setUint8()`

> **Note**
>
> All Dataview ES 6 prototype methods are new in JavaScript runtime 2.0.

Symbol

The following ES 6 methods are supported:

- `Symbol.for()`
- `Symbol.keyFor()`

> **Note**
>
> All Symbol ES 6 methods are new in JavaScript runtime 2.0.

Text Decoder

The following prototype methods are supported:

- `TextDecoder.prototype.decode()`

The following prototype accessor properties are supported:

- `TextDecoder.prototype.encoding`
- `TextDecoder.prototype.fatal`
- `TextDecoder.prototype.ignoreBOM`

Text Encoder

The following prototype methods are supported:

- `TextEncoder.prototype.encode()`
- `TextEncoder.prototype.encodeInto()`

Console

This is a helper object for debugging. It only supports the `log()` method, to record log messages.

> **Note**
>
> CloudFront Functions doesn't support comma syntax, such as `console.log('a', 'b')`. Instead, use the `console.log('a' + ' ' + 'b')` format.

Error types

The following error objects are supported:

- `Error`
- `EvalError`
- `InternalError`
- `RangeError`
- `ReferenceError`
- `SyntaxError`
- `TypeError`
- `URIError`

Globals

The `globalThis` object is supported.

The following ES 5.1 global functions are supported:

- `decodeURI()`
- `decodeURIComponent()`
- `encodeURI()`
- `encodeURIComponent()`
- `isFinite()`
- `isNaN()`
- `parseFloat()`
- `parseInt()`

The following ES 6 global functions are supported:

- `atob()`
- `btoa()`

> **Note**
>
> `atob()` and `btoa()` are new in JavaScript runtime 2.0.

The following global constants are supported:

- `NaN`
- `Infinity`
- `undefined`
- `arguments`

Built-in modules

The following built-in modules are supported.

Modules

- Buffer
- Query string
- Crypto

Buffer

The module provides the following methods:

- `Buffer.alloc(size[, fill[, encoding]])`

  Allocate a `Buffer`.

  - `size`: Buffer size. Enter an integer.
  - `fill`: Optional. Enter a string, `Buffer`, Uint8Array, or integer. Default is `0`.
  - `encoding`: Optional. When `fill` is a string, enter one of the following: `utf8`, `hex`, `base64`, `base64url`. Default is `utf8`.
- `Buffer.allocUnsafe(size)`

Allocate a non-initialized Buffer.

- size: Enter an integer.

- Buffer.byteLength(value[, encoding])

  Return the length of a value, in bytes.

  - value: A string, Buffer, TypedArray, Dataview, or Arraybuffer.
  - encoding: Optional. When value is a string, enter one of the following: utf8, hex, base64, base64url. Default is utf8.

- Buffer.compare(buffer1, buffer2)

  Compare two Buffers to help sort arrays. Returns 0 if they're the same, -1 if buffer1 comes first, or 1 if buffer2 comes first.

  - buffer1: Enter a Buffer.
  - buffer2: Enter a different Buffer.

- Buffer.concat(list[, totalLength])

  Concatenate multiple Buffers. Returns 0 if none. Returns up to totalLength.

  - list: Enter a list of Buffers. Note this will be truncated to totalLength.
  - totalLength: Optional. Enter an unsigned integer. Use sum of Buffer instances in list if blank.

- Buffer.from(array)

  Create a Buffer from an array.

  - array: Enter a byte array from 0 to 255.

- Buffer.from(arrayBuffer, byteOffset[, length]))

  Create a view from arrayBuffer, starting at offset byteOffset with length length.

  - arrayBuffer: Enter a Buffer array.
  - byteOffset: Enter an integer.
  - length: Optional. Enter an integer.

- Buffer.from(buffer)

  Create a copy of the Buffer.

  - buffer: Enter a Buffer.

- `Buffer.from(object[, offsetOrEncoding[, length]])`

  Create a `Buffer` from an object. Returns `Buffer.from(object.valueOf(), offsetOrEncoding, length)` if `valueOf()` is not equal to the object.

  - `object`: Enter an object.
  - `offsetOrEncoding`: Optional. Enter an integer or encoding string.
  - `length`: Optional. Enter an integer.
- `Buffer.from(string[, encoding])`

  Create a `Buffer` from a string.

  - `string`: Enter a string.
  - `encoding`: Optional. Enter one of the following: `utf8`, `hex`, `base64`, `base64url`. Default is `utf8`.
- `Buffer.isBuffer(object)`

  Check if `object` is a Buffer. Returns `true` or `false`.

  - `object`: Enter an object.
- `Buffer.isEncoding(encoding)`

  Check if `encoding` is supported. Returns `true` or `false`.

  - `encoding`: Optional. Enter one of the following: `utf8`, `hex`, `base64`, `base64url`. Default is `utf8`.

The module provides the following buffer prototype methods:

- `Buffer.prototype.compare(target[, targetStart[, targetEnd[, sourceStart[, sourceEnd]]]])`

  Compare `Buffer` with target. Returns `0` if they're the same, `1` if `buffer` comes first, or `-1` if `target` comes first.

  - `target`: Enter a `Buffer`.
  - `targetStart`: Optional. Enter an integer. Default is 0.
  - `targetEnd`: Optional. Enter an integer. Default is `target` length.
  - `sourceStart`: Optional. Enter an integer. Default is 0.
  - `sourceEnd`: Optional. Enter an integer. Default is `Buffer` length.

- `Buffer.prototype.copy(target[, targetStart[, sourceStart[, sourceEnd]]])`

  Copy buffer to `target`.

  - `target`: Enter a `Buffer` or `Uint8Array`.
  - `targetStart`: Optional. Enter an integer. Default is 0.
  - `sourceStart`: Optional. Enter an integer. Default is 0.
  - `sourceEnd`: Optional. Enter an integer. Default is `Buffer` length.
- `Buffer.prototype.equals(otherBuffer)`

  Compare `Buffer` to `otherBuffer`. Returns `true` or `false`.

  - `otherBuffer`: Enter a string.
- `Buffer.prototype.fill(value[, offset[, end][, encoding])`

  Fill `Buffer` with `value`.

  - `value`: Enter a string, `Buffer`, or integer.
  - `offset`: Optional. Enter an integer.
  - end: Optional. Enter an integer.
  - `encoding`: Optional. Enter one of the following: `utf8`, `hex`, `base64`, `base64url`. Default is `utf8`.
- `Buffer.prototype.includes(value[, byteOffset][, encoding])`

  Search for `value` in `Buffer`. Returns `true` or `false`.

  - `value`: Enter a string, `Buffer`, `Uint8Array`, or integer.
  - `byteOffset`: Optional. Enter an integer.
  - `encoding`: Optional. Enter one of the following: `utf8`, `hex`, `base64`, `base64url`. Default is `utf8`.
- `Buffer.prototype.indexOf(value[, byteOffset][, encoding])`

  Search for first `value` in `Buffer`. Returns `index` if found; returns `-1` if not found.

  - `value`: Enter a string, `Buffer`, Uint8Array, or integer from 0 to 255.
  - `byteOffset`: Optional. Enter an integer.
  - `encoding`: Optional. Enter one of the following if `value` is a string: `utf8`, `hex`, `base64`, `base64url`. Default is `utf8`.
- `Buffer.prototype.lastIndexOf(value[, byteOffset][, encoding])`

Search for last `value` in `Buffer`. Returns `index` if found; returns `-1` if not found.

- `value`: Enter a string, `Buffer`, Uint8Array, or integer from 0 to 255.
- `byteOffset`: Optional. Enter an integer.
- `encoding`: Optional. Enter one of the following if `value` is a string: `utf8`, `hex`, `base64`, `base64url`. Default is `utf8`.

- `Buffer.prototype.readInt8(offset)`

  Read `Int8` at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readIntBE(offset, byteLength)`

  Read `Int` as big-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
  - `byteLength`: Optional. Enter an integer from 1 to 6.
- `Buffer.prototype.readInt16BE(offset)`

  Read `Int16` as big-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readInt32BE(offset)`

  Read `Int32` as big-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readIntLE(offset, byteLength)`

  Read `Int` as little-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.readInt16LE(offset)`

  Read `Int16` as little-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readInt32LE(offset)`

  Read `Int32` as little-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readUInt8(offset)`

  Read `UInt8` at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readUIntBE(offset, byteLength)`

  Read `UInt` as big-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.readUInt16BE(offset)`

  Read `UInt16` as big-endian at `offset` from `Buffer`.

- • `offset`: Enter an integer.
- `Buffer.prototype.readUInt32BE(offset)`

  Read `UInt32` as big-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readUIntLE(offset, byteLength)`

  Read `UInt` as little-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.readUInt16LE(offset)`

  Read `UInt16` as little-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readUInt32LE(offset)`

  Read `UInt32` as little-endian at `offset` from `Buffer`.

  - `offset`: Enter an integer.
- `Buffer.prototype.readDoubleBE([offset])`

  Read a 64-bit double as big-endian at `offset` from `Buffer`.

  - `offset`: Optional. Enter an integer.

- `Buffer.prototype.readDoubleLE([offset])`

  Read a 64-bit double as little-endian at `offset` from `Buffer`.

  - `offset`: Optional. Enter an integer.
- `Buffer.prototype.readFloatBE([offset])`

  Read a 32-bit float as big-endian at `offset` from `Buffer`.

  - `offset`: Optional. Enter an integer.
- `Buffer.prototype.readFloatLE([offset])`

  Read a 32-bit float as little-endian at `offset` from `Buffer`.

  - `offset`: Optional. Enter an integer.
- `Buffer.prototype.subarray([start[, end]])`

  Returns a copy of `Buffer` that is offset and cropped with a new `start` and `end`.

  - `start`: Optional. Enter an integer. Default is 0.
  - `end`: Optional. Enter an integer. Default is buffer length.
- `Buffer.prototype.swap16()`

  Swap the `Buffer` array byte order, treating it as an array of 16-bit numbers. `Buffer` length must be divisible by 2, or you will receive an error.
- `Buffer.prototype.swap32()`

  Swap the `Buffer` array byte order, treating it as an array of 32-bit numbers. `Buffer` length must be divisible by 4, or you will receive an error.
- `Buffer.prototype.swap64()`

  Swap the `Buffer` array byte order, treating it as an array of 64-bit numbers. `Buffer` length must be divisible by 8, or you will receive an error.
- `Buffer.prototype.toJSON()`

  Returns `Buffer` as a JSON.
- `Buffer.prototype.toString([encoding[, start[, end]]])`

  Convert `Buffer`, from `start` to `end`, to encoded string.

  - encoding: Optional. Enter one of the following: utf8, hex, base64, or base64url. Default is utf8.
  - start: Optional. Enter an integer. Default is 0.
  - end: Optional. Enter an integer. Default is buffer length.
- Buffer.prototype.write(string[, offset[, length]][, encoding])

  Write encoded string to Buffer if there is space, or a truncated string if there is not enough space.
  - string: Enter a string.
  - offset: Optional. Enter an integer. Default is 0.
  - length: Optional. Enter an integer. Default is the length of the string.
  - encoding: Optional. Optionally enter one of the following: utf8, hex, base64, or base64url. Default is utf8.
- Buffer.prototype.writeInt8(value, offset, byteLength)

  Write Int8 value of byteLength at offset to Buffer.
  - value: Enter an integer.
  - offset: Enter an integer
  - byteLength: Enter an integer from 1 to 6.
- Buffer.prototype.writeIntBE(value, offset, byteLength)

  Write value at offset to Buffer, using big-endian.
  - value: Enter an integer.
  - offset: Enter an integer
  - byteLength: Enter an integer from 1 to 6.
- Buffer.prototype.writeInt16BE(value, offset, byteLength)

  Write value at offset to Buffer, using big-endian.
  - value: Enter an integer.
  - offset: Enter an integer
  - byteLength: Enter an integer from 1 to 6.
- Buffer.prototype.writeInt32BE(value, offset, byteLength)

  Write value at offset to Buffer, using big-endian.

  - `value`: Enter an integer.
  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeIntLE(offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using little-endian.
  - `offset`: Enter an integer.
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeInt16LE(offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using little-endian.
  - `offset`: Enter an integer.
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeInt32LE(offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using little-endian.
  - `offset`: Enter an integer.
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeUInt8(value, offset, byteLength)`

  Write `UInt8` `value` of `byteLength` at `offset` to `Buffer`.
  - `value`: Enter an integer.
  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeUIntBE(value, offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using big-endian.
  - `value`: Enter an integer.
  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeUInt16BE(value, offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using big-endian.
  - `value`: Enter an integer.

  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeUInt32BE(value, offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using big-endian.
  - `value`: Enter an integer.
  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeUIntLE(value, offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using little-endian.
  - `value`: Enter an integer.
  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeUInt16LE(value, offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using little-endian.
  - `value`: Enter an integer.
  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeUInt32LE(value, offset, byteLength)`

  Write `value` at `offset` to `Buffer`, using little-endian.
  - `value`: Enter an integer.
  - `offset`: Enter an integer
  - `byteLength`: Enter an integer from 1 to 6.
- `Buffer.prototype.writeDoubleBE(value, [offset])`

  Write `value` at `offset` to `Buffer`, using big-endian.
  - `value`: Enter an integer.
  - `offset`: Optional. Enter an integer. Default is 0.
- `Buffer.prototype.writeDoubleLE(value, [offset])`

  Write `value` at `offset` to `Buffer`, using little-endian.

  - `value`: Enter an integer.
  - `offset`: Optional. Enter an integer. Default is 0.
- `Buffer.prototype.writeFloatBE(value, [offset])`

  Write `value` at `offset` to `Buffer`, using big-endian.
  - `value`: Enter an integer.
  - `offset`: Optional. Enter an integer. Default is 0.
- `Buffer.prototype.writeFloatLE(value, [offset])`

  Write `value` at `offset` to `Buffer`, using little-endian.
  - `value`: Enter an integer.
  - `offset`: Optional. Enter an integer. Default is 0.

The following instance methods are supported:

- `buffer[index]`

  Get and set octet (byte) at `index` in `Buffer`.
  - Get a number from `0` to `255`. Or set a number from `0` to `255`.

The following instance properties are supported:

- `buffer`

  Get the `ArrayBuffer` object for the buffer.
- `byteOffset`

  Get the `byteOffset` of the buffer's `Arraybuffer` object.
- `length`

  Get the buffer byte count.

> **Note**
>
> All Buffer module methods are new in JavaScript runtime 2.0.

Query string

> **Note**
>
> The CloudFront Functions event object automatically parses URL query strings for you. That means that in most cases you don't need to use this module.

The query string module (`querystring`) provides methods for parsing and formatting URL query strings. You can load the module using `require('querystring')`. The module provides the following methods.

`querystring.escape(`*`string`*`)`

URL-encodes the given `string`, returning an escaped query string. The method is used by `querystring.stringify()` and should not be used directly.

`querystring.parse(`*`string`*`[, `*`separator`*`[, `*`equal`*`[, `*`options`*`]]])`

Parses a query string (`string`) and returns an object.

The `separator` parameter is a substring for delimiting key and value pairs in the query string. By default it is &.

The `equal` parameter is a substring for delimiting keys and values in the query string. By default it is =.

The `options` parameter is an object with the following keys:

`decodeURIComponent` *`function`*

A function to decode percent-encoded characters in the query string. By default it is `querystring.unescape()`.

`maxKeys` *`number`*

The maximum number of keys to parse. By default it is `1000`. Use a value of `0` to remove the limitations for counting keys.

By default, percent-encoded characters within the query string are assumed to use the UTF-8 encoding. Invalid UTF-8 sequences are replaced with the `U+FFFD` replacement character.

For example, for the following query string:

```
'name=value&abc=xyz&abc=123'
```

The return value of `querystring.parse()` is:

```
{
name: 'value',
abc: ['xyz', '123']
}
```

`querystring.decode()` is an alias for `querystring.parse()`.

`querystring.stringify(`*`object`*`[, `*`separator`*`[, `*`equal`*`[, `*`options`*`]]])`

Serializes an `object` and returns a query string.

The `separator` parameter is a substring for delimiting key and value pairs in the query string. By default it is &.

The `equal` parameter is a substring for delimiting keys and values in the query string. By default it is =.

The `options` parameter is an object with the following keys:

`encodeURIComponent` *`function`*

The function to use for converting URL-unsafe characters to percent-encoding in the query string. By default it is `querystring.escape()`.

By default, characters that require percent-encoding within the query string are encoded as UTF-8. To use a different encoding, specify the `encodeURIComponent` option.

For example, for the following code:

```
querystring.stringify({ name: 'value', abc: ['xyz', '123'], anotherName: '' });
```

The return value is:

```
'name=value&abc=xyz&abc=123&anotherName='
```

`querystring.encode()` is an alias for `querystring.stringify()`.

`querystring.unescape(`*`string`*`)`

Decodes URL percent-encoded characters in the given `string`, returning an unescaped query string. This method is used by `querystring.parse()` and should not be used directly.

Crypto

The cryptographic module (`crypto`) provides standard hashing and hash-based message authentication code (HMAC) helpers. You can load the module using `require('crypto')`.

Hashing methods

`crypto.createHash(`*`algorithm`*`)`

Creates and returns a hash object that you can use to generate hash digests using the given algorithm: `md5`, `sha1`, or `sha256`.

`hash.update(`*`data`*`)`

Updates the hash content with the given `data`.

`hash.digest([`*`encoding`*`])`

Calculates the digest of all of the data passed using `hash.update()`. The encoding can be `hex`, `base64`, or `base64url`.

HMAC methods

`crypto.createHmac(`*`algorithm, secret key`*`)`

Creates and returns an HMAC object that uses the given `algorithm` and `secret key`. The algorithm can be `md5`, `sha1`, or `sha256`.

`hmac.update(`*`data`*`)`

Updates the HMAC content with the given `data`.

`hmac.digest([`*`encoding`*`])`

Calculates the digest of all of the data passed using `hmac.update()`. The encoding can be `hex`, `base64`, or `base64url`.

Restricted features

The following JavaScript language features are either unsupported or restricted due to security concerns.

Dynamic code evaluation

Dynamic code evaluation is not supported. Both `eval()` and `Function` constructors throw an error if attempted. For example, `const sum = new Function('a', 'b', 'return a + b')` throws an error.

Timers

The `setTimeout()`, `setImmediate()`, and `clearTimeout()` functions are not supported. There is no provision to defer or yield within a function run. Your function must synchronously run to completion.

Date and timestamps

For security reasons, there is no access to high-resolution timers. All `Date` methods to query the current time always return the same value during the lifetime of a single function run. The returned timestamp is the time when the function started running. Consequently, you cannot measure elapsed time in your function.

File system access

There is no file system access. For example, there is no `fs` module for file system access like there is in Node.js.

Process access

There is no process access. For example, there is no `process` global object for processing information access like there is in Node.js.

Environment variables

There is no access to environment variables. Instead, you can use CloudFront KeyValueStore to create a centralized datastore of key-value pairs for your CloudFront Functions. CloudFront KeyValueStore enables dynamic updates to your configuration data without needing to deploy code changes. For more information, see Amazon CloudFront KeyValueStore.

Network access

Network calls are supported only to an origin that is configured on the same distribution. A function can make one network call per invocation. For more information, see Helper methods for network calls.

Helper methods for key value stores

> **Note**
>
> Key value store helper method calls from CloudFront Functions don't trigger an AWS CloudTrail data event. These events aren't logged in the CloudTrail event history. For more information, see Logging Amazon CloudFront API calls using AWS CloudTrail.

This section applies if you use the CloudFront Key Value Store to include key values in the function that you create. CloudFront Functions has a module that provides three helper methods to read values from the key value store.

To use this module in the function code, make sure that you have associated a key value store with the function.

Next, include the following statements in the first lines of the function code:

```
import cf from 'cloudfront';
const kvsHandle = cf.kvs();
```

`get()` method

Use this method to return the key value for the key name that you specify.

Request

```
get("key", options);
```

- `key`: The name of the key whose value needs to be fetched
- `options`: There is one option, `format`. It makes sure that the function parses the data correctly. Possible values:

- `string`: (Default) UTF8 encoded
- `json`
- `bytes`: Raw binary data buffer

Request example

```
const value = await kvsHandle.get("myFunctionKey", { format: "string"});
```

Response

The response is a `promise` that resolves to a value in the format requested by using `options`. By default, the value is returned as a string.

Error handling

The `get()` method will return an error when the key that you requested doesn't exist in the associated key value store. To manage this use case, you can add a `try` and `catch` block to your code.

Warning

Using promise combinators (for example, `Promise.all`, `Promise.any`, and promise chain methods (for example, `then` and `catch`) can require high function memory usage. If your function exceeds the maximum function memory quota, it will fail to execute. To avoid this error, we recommend that you use the `await` syntax sequentially or in loops to request multiple values.

Example

```
var value1 = await kvs.get('key1');
var value2 = await kvs.get('key2');
```

Currently, using promise combinators to get multiple values won't improve performance, such as the following example.

```
var values = await Promise.all([kvs.get('key1'), kvs.get('key2'),]);
```

`exists()` method

Use this method to identify whether or not the key exists in the key value store.

Request

```
exists("key");
```

Request example

```
const exist = await kvsHandle.exists("myFunctionkey");
```

Response

The response is a `promise` that returns a Boolean (`true` or `false`). This value specifies whether or not the key exists in the key value store.

`meta()` method

Use this method to return metadata about the key value store.

Request

```
meta();
```

Request example

```
const meta = await kvsHandle.meta();
```

Response

The response is a `promise` that resolves to an object with the following properties:

- `creationDateTime`: The date and time that the key value store was created, in ISO 8601 format.
- `lastUpdatedDateTime`: The date and time that the key value store was last synced from the source, in ISO 8601 format. The value doesn't include the propagation time to the edge.
- `keyCount`: The total number of keys in the KVS after the last sync from the source.

Response example

```
{keyCount:3,creationDateTime:2023-11-30T23:07:55.765Z,lastUpdatedDateTime:2023-12-15T03:57:52.4
```

Helper methods for origin modification

This section applies if you dynamically update or change the origin used on the request inside your CloudFront Functions code. You can update the origin on *viewer request* CloudFront Functions only. CloudFront Functions has a module that provides helper methods to dynamically update or change the origin.

To use this module, create a CloudFront function using JavaScript runtime 2.0 and include the following statement in the first line of the function code:

```
import cf from 'cloudfront';
```

For more information, see JavaScript runtime 2.0 features for CloudFront Functions.

> **Note**
>
> The Test API and Test console pages don't test whether an origin modification has occurred. However, testing makes sure that the function code executes without error.

Choose between CloudFront Functions and Lambda@Edge

You can update your origins by using either CloudFront Functions or Lambda@Edge.

When using CloudFront Functions to update origins, you use the *viewer request* event trigger, which means this logic will run on every request when this function is used. When using Lambda@Edge, the origin updating capabilities are on the *origin request* event trigger, which means this logic only runs on cache misses.

Your choice depends largely on your workload and any existing usage of CloudFront Functions and Lambda@Edge on your distributions. The following considerations can help you decide whether to use CloudFront Functions or Lambda@Edge to update your origins.

CloudFront Functions is most useful in the following situations:

- When your requests are dynamic (meaning they cannot be cached) and will always go to origin. CloudFront Functions provides better performance and lower overall cost.

- When you already have an existing viewer request CloudFront function that will run on every request, you can add the origin updating logic into the existing function.

To use CloudFront Functions to update origins, see the helper methods in the following topics.

Lambda@Edge is most useful in the following situations:

- When you have highly cacheable content, Lambda@Edge can be more cost-efficient because it runs only on cache misses, while CloudFront Functions runs on every request.
- When you already have an existing origin request Lambda@Edge function, you can add the origin updating logic into the existing function.
- When your origin update logic requires fetching data from third-party data sources, such as Amazon DynamoDB or Amazon S3.

For more information about Lambda@Edge, see Customize at the edge with Lambda@Edge.

updateRequestOrigin() method

Use the `updateRequestOrigin()` method to update the origin settings for a request. You can use this method to update existing origin properties for origins that are already defined in your distribution, or to define a new origin for the request. To do so, specify the properties that you want to change.

Important

Any settings that you don't specify in the `updateRequestOrigin()` will inherit the *same settings* from the existing origin's configuration.

The origin set by the `updateRequestOrigin()` method can be any HTTP endpoint and doesn't need to be an existing origin within your CloudFront distribution.

Notes

- If you're updating an origin that is part of an origin group, only the *primary origin* of the origin group is updated. The secondary origin remains unchanged. Any response code from the modified origin that matches the failover criteria will trigger a failover to the secondary origin.

- If you are changing the origin type and have OAC enabled, make sure that the origin type in `originAccessControlConfig` matches the new origin type.
- You can't use the `updateRequestOrigin()` method to update VPC origins. The request will fail.

Request

```
updateRequestOrigin({origin properties})
```

The `origin properties` can contain the following:

domainName (optional)

The domain name of the origin. If this is not provided, the domain name from the assigned origin is used instead.

For custom origins

Specify a DNS domain name, such as `www.example.com`. The domain name can't include a colon (`:`) and can't be an IP address. The domain name can be up to 253 characters.

For S3 origins

Specify the DNS domain name of the Amazon S3 bucket, such as `amzn-s3-demo-bucket.s3.eu-west-1.amazonaws.com`. The name can be up to 128 characters, and must be all lowercase.

hostHeader (optional, for non-S3 custom origins)

The host header to use when making the request to the origin. If this is not provided, the value from the domainName parameter is used. If neither host header or domain name parameter are provided, the domain name from the assigned origin is used or the host header from the incoming request if the forward to origin (FTO) policy includes the host. The host header can't include a colon (`:`) and can't be an IP address. The host header can be up to 253 characters.

originPath (optional)

The directory path at the origin where the request should locate content. The path should start with a forward slash (/) but shouldn't end with one. For example, it shouldn't end with `example-path/`. If this is not provided, then the origin path from the assigned origin is used.

For custom origins

The path should be URL encoded and have a maximum length of 255 characters.

customHeaders (optional)

You can include custom headers with the request by specifying a header name and value pair for each custom header. The format is different than that of the request and response headers in the event structure. Use the following key-value pair syntax:

```
{"key1": "value1", "key2": "value2", ...}
```

You can't add headers that are disallowed, and a header with the same name can't also be present in the incoming request `headers`. Header name must be lowercase in your function code. When CloudFront Functions converts the event object back into an HTTP request, the first letter of each word in header names is capitalized, and words are separated by a hyphen.

For example, if your function code adds a header named `example-header-name`, CloudFront converts this to `Example-Header-Name` in the HTTP request. For more information, see Custom headers that CloudFront can't add to origin requests and Restrictions on edge functions.

If this is not provided, then any custom headers from the assigned origin are used.

connectionAttempts (optional)

The number of times that CloudFront attempts to connect to the origin. The minimum is 1 and the maximum is 3. If this is not provided, the connection attempts from the assigned origin are used.

originShield (optional)

This enables or updates CloudFront Origin Shield. Using Origin Shield can help reduce the load on your origin. For more information, see Use Amazon CloudFront Origin Shield. If this is not provided, the Origin Shield settings from the assigned origin are used.

enabled (required)

Boolean expression to enable or disable Origin Shield. Accepts a `true` or `false` value.

region (required when enabled)

The AWS Region for Origin Shield. Specify the AWS Region that has the lowest latency to your origin. Use the Region code, not the Region name. For example, use `us-east-2` to specify the US East (Ohio) Region.

When you enable CloudFront Origin Shield, you must specify the AWS Region for it. For a list of available AWS Regions and help choosing the best Region for your origin, see Choose the AWS Region for Origin Shield.

originAccessControlConfig (optional)

The unique identifier of an origin access control (OAC) for this origin. This is only used when the origin supports a CloudFront OAC, such as Amazon S3, Lambda function URLs, Lambda Web Functions, MediaStore, and MediaPackage V2. If this is not provided, then the OAC settings from the assigned origin are used.

This does not support the legacy origin access identity (OAI). For more information, see Restrict access to an AWS origin.

enabled (required)

Boolean expression to enable or disable OAC. Accepts a `true` or `false` value.

signingBehavior (required when enabled)

Specifies which requests CloudFront signs (adds authentication information to). Specify `always` for the most common use case. For more information, see Advanced settings for origin access control.

This field can have one of the following values:

- `always` – CloudFront signs all origin requests, overwriting the `Authorization` header from the viewer request if one exists.
- `never` – CloudFront doesn't sign any origin requests. This value turns off origin access control for the origin.
- `no-override` – If the viewer request doesn't contain the `Authorization` header, then CloudFront signs the origin request. If the viewer request contains the `Authorization` header, then CloudFront doesn't sign the origin request and instead passes along the `Authorization` header from the viewer request.

> **Warning**
>
> To pass along the `Authorization` header from the viewer request, you must add it to an origin request policy for all cache behaviors that use origins associated with this origin access control. For more information, see Control origin requests with a policy.

- `always-amz-auth` – CloudFront signs all origin requests, but it doesn't replace the `Authorization` header from the viewer request. CloudFront adds its signature in a dedicated CloudFront header and passes the viewer's `Authorization` header to the origin unmodified. This value is required for Lambda Web Function origins and is not supported for any other origin type. For more information, see Signing behavior.

signingProtocol (required when enabled)

The signing protocol of the OAC, which determines how CloudFront signs (authenticates) requests. Specify `sigv4` for most origins. Specify `sigv4a` for an Amazon S3 Multi-Region Access Point origin or a global Lambda Web Function origin.

originType (required when enabled)

The type of origin for this OAC. Valid values include `s3`, `mediapackagev2`, `mediastore`, and `lambda`.

timeouts (optional)

Timeouts that you can specify for how long CloudFront should attempt to wait for origins to respond or send data. If this is not provided, then the timeout settings from the assigned origin are used.

> **Note**
>
> Unless specified, these timeouts support both custom origins and Amazon S3 origins.

readTimeout (optional)

The `readTimeout` applies to both of the following values:

- How long (in seconds) CloudFront waits for a response after forwarding a request to the origin.

- How long (in seconds) CloudFront waits after receiving a packet of a response from the origin and before receiving the next packet.

The minimum timeout is 1 second and the maximum is 120 seconds. For more information, see Response timeout.

responseCompletionTimeout (optional)

The time (in seconds) that a request from CloudFront to the origin can stay open and wait for a response. If the complete response isn't received from the origin by this time, CloudFront ends the connection.

The value for `responseCompletionTimeout` must be equal to or greater than the value for the `readTimeout`. For more information, see Response completion timeout.

keepAliveTimeout (optional)

This timeout only applies to custom origins, not Amazon S3 origins. (S3 origin configurations will ignore these settings.)

The `keepAliveTimeout` specifies how long CloudFront should try to maintain the connection to the origin after receiving the last packet of the response. The minimum timeout is 1 second and the maximum is 120 seconds. For more information, see Keep-alive timeout (custom and VPC origins only).

connectionTimeout (optional)

The number of seconds that CloudFront waits when trying to establish a connection to the origin. The minimum timeout is 1 second and the maximum is 10 seconds. For more information, see Connection timeout.

customOriginConfig (optional)

Use `customOriginConfig` to specify connection settings for origins that are *not* an Amazon S3 bucket. There is one exception: you can specify these settings if the S3 bucket is configured with static website hosting. (Other types of S3 bucket configurations will ignore these settings.) If `customOriginConfig` is not provided, then the settings from the assigned origin are used.

port (required)

The HTTP port that CloudFront uses to connect to the origin. Specify the HTTP port that the origin listens on.

protocol (required)

Specifies the protocol (HTTP or HTTPS) that CloudFront uses to connect to the origin. Valid values are as follows:

- `http` – CloudFront always uses HTTP to connect to the origin
- `https` – CloudFront always uses HTTPS to connect to the origin

sslProtocols (required)

A list that specifies the minimum SSL/TLS protocol that CloudFront uses when connecting to your origin over HTTPS. Valid values include `SSLv3`, `TLSv1`, `TLSv1.1`, and `TLSv1.2`. For more information, see Minimum origin SSL protocol.

ipAddressType (optional)

Specifies the IP address type that CloudFront uses to connect to the origin. Valid values include `ipv4`, `ipv6`, and `dualstack`. Changing `ipAddressType` is only supported when the `domainName` property is also being changed.

sni (optional, for non-S3 custom origins)

The Server Name Indication (SNI) is an extension to the Transport Layer Security (TLS) protocol by which a client indicates which hostname it's attempting to connect to at the start of the TLS handshake process. This value should match a common name on a TLS certificate on your origin server. Otherwise, your origin server might throw an error.

If this is not provided, the value from the `hostHeader` parameter is used. If the host header is not provided, the value from the `domainName` parameter is used.

If neither host header or domain name parameter are provided, the domain name from the assigned origin is used or the host header from the incoming request if the forward to origin (FTO) policy includes the host. The SNI can't include a colon (`:`) and can't be an IP address. The SNI can be up to 253 characters.

allowedCertificateNames (optional, for non-S3 custom origins)

You can include a list of valid certificate names to be used by CloudFront to validate the domain matching from your origin server TLS certificate during the TLS handshake with your origin server. This field expects an array of valid domain names and can include wildcard domains, such as `*.example.com`.

You can specify up to 20 allowed certificate names. Each certificate name can have up to 64 characters.

Example– Update to Amazon S3 request origin

The following example changes the viewer request's origin to an S3 bucket, enables OAC, and resets custom headers sent to the origin.

```
cf.updateRequestOrigin({
    "domainName" : "amzn-s3-demo-bucket-in-us-east-1.s3.us-east-1.amazonaws.com",
    "originAccessControlConfig": {
        "enabled": true,
        "signingBehavior": "always",
        "signingProtocol": "sigv4",
        "originType": "s3"
    },
    // Empty object resets any header configured on the assigned origin
    "customHeaders": {}
});
```

Example– Update to Application Load Balancer request origin

The following example changes the viewer request's origin to an Application Load Balancer origin and sets a custom header and timeouts.

```
cf.updateRequestOrigin({
    "domainName" : "example-1234567890.us-east-1.elb.amazonaws.com",
    "timeouts": {
        "readTimeout": 30,
        "connectionTimeout": 5
    },
    "customHeaders": {
        "x-stage": "production",
        "x-region": "us-east-1"
    }
});
```

Example– Update to origin with Origin Shield enabled

In the following example, the origin in the distribution has Origin Shield enabled. The function code updates only the domain name used for the origin and omits all the other optional parameters. In this case, Origin Shield will still be used with the modified origin domain name because the Origin Shield parameters were not updated.

```
cf.updateRequestOrigin({
```

```
    "domainName" : "www.example.com"
});
```

Example– Update the host header, SNI, and allowed certificate names

> **Warning**
>
> For most use cases, you won't need to use this type of modification to requests going to your origin. These parameters shouldn't be used unless you understand the impact of changing these values.

The following example changes the domain name, the host header, SNI, and allowed certificates on the request to the origin.

```
cf.updateRequestOrigin({
    "domainName": "www.example.com",
    "hostHeader": "test.example.com",
    "sni": "test.example.net",
    "allowedCertificateNames": ["*.example.com", "*.example.net"],
});
```

selectRequestOriginById() method

Use `selectRequestOriginById()` to update an existing origin by selecting a different origin that's already configured in your distribution. This method uses all the same settings that are defined by the updated origin.

This method only accepts origins that are already defined in the same distribution used when running the function. Origins are referenced by the origin ID, which is the origin name that you defined when setting up the origin.

If you have a VPC origin configured in your distribution, you can use this method to update your origin to your VPC origin. For more information, see Restrict access with VPC origins.

If you have an origin with mutual TLS (mTLS) enabled in your distribution, you can use this method to update your origin to your mTLS-enabled origin. For more information, see Using CloudFront Functions with origin mutual TLS.

Request

```
cf.selectRequestOriginById(origin_id, {origin_overrides})
```

In the previous example, `origin_id` is a string that points to the origin name of an origin in the distribution that's running the function. The `origin_overrides` parameter can contain the following:

hostHeader (optional, for non-S3 custom origins)

The host header to use when making the request to the origin. If this is not provided, the value from the `domainName` parameter is used.

If neither host header or domain name parameter are provided, the domain name from the assigned origin is used or the host header from the incoming request if the forward to origin (FTO) policy includes the host. The host header can't include a colon (`:`) and can't be an IP address. The host header can be up to 253 characters.

sni (optional, for non-S3 custom origins)

The Server Name Indication (SNI) is an extension to the Transport Layer Security (TLS) protocol by which a client indicates which hostname it's attempting to connect to at the start of the TLS handshake process. This value should match a common name on a TLS certificate on your origin server. Otherwise, your origin server might throw an error.

If this is not provided, the value from the `hostHeader` parameter is used. If the host header is not provided, the value from the `domainName` parameter is used.

If neither host header or domain name parameter are provided, the domain name from the assigned origin is used or the host header from the incoming request if the forward to origin (FTO) policy includes the host. The SNI can't include a colon (`:`) and can't be an IP address. The SNI can be up to 253 characters.

allowedCertificateNames (optional, for non-S3 custom origins)

You can include a list of valid certificate names to be used by CloudFront to validate the domain matching from your origin server TLS certificate during the TLS handshake with your origin server. This field expects an array of valid domain names and can include wildcard domains, such as `*.example.com`.

You can specify up to 20 allowed certificate names. Each certificate name can have up to 64 characters.

Request

```
selectRequestOriginById(origin_id)
```

In the preceding example, `origin_id` is a string that points to the origin name of an origin in the distribution that's running the function.

Example– Select Amazon S3 request origin

The following example selects the origin named `amzn-s3-demo-bucket-in-us-east-1` from the list of origins associated with the distribution, and applies the configuration settings of the `amzn-s3-demo-bucket-in-us-east-1` origin to the request.

```
cf.selectRequestOriginById("amzn-s3-demo-bucket-in-us-east-1");
```

Example– Select Application Load Balancer request origin

The following example selects an Application Load Balancer origin named `myALB-prod` from the list of origins associated with the distribution, and applies the configuration settings of `myALB-prod` to the request.

```
cf.selectRequestOriginById("myALB-prod");
```

Example– Select Application Load Balancer request origin and override the host header

Like the previous example, the following example selects an Application Load Balancer origin named `myALB-prod` from the list of origins associated with the distribution, and applies the configuration settings of `myALB-prod` to the request. However, this example overrides the host header value using `origin_overrides`.

```
cf.overrideRequestOrigin("myALB-prod",{
        "hostHeader" : "test.example.com"
});
```

createRequestOriginGroup() method

Use `createRequestOriginGroup()` to define two origins to use as an origin group for failover in scenarios that require high availability.

An origin group includes two origins (a primary and a secondary) and a failover criteria that you specify. You create an origin group to support origin failover in CloudFront. When you create or update an origin group using this method, you can specify the origin group instead of a single origin. CloudFront will failover from the primary origin to the secondary origin, using the failover criteria.

If you have a VPC origin configured in your distribution, you can use this method to create an origin group using a VPC origin. For more information, see Restrict access with VPC origins.

If you have an origin with mutual TLS (mTLS) enabled in your distribution, you can use this method to create an origin group that includes your mTLS-enabled origin. For more information, see Using CloudFront Functions with origin mutual TLS.

Request

```
createRequestOriginGroup({origin_group_properties})
```

In the preceding example, the `origin_group_properties` can contain the following:

originIds (required)

Array of `origin_ids`, where the `origin_id` is a string that points to the origin name of an origin in the distribution running the function. You must provide two origins as part of the array. The first origin in the list is the primary origin and the second serves as the second origin for failover purposes.

originOverrides (optional)

A few advanced settings are allowed to be overwritten by using the `{origin_overrides}` parameter. The `origin overrides` can contain the following:

hostHeader (optional, for non-S3 custom origins)

The host header to use when making the request to the origin. If this is not provided, the value from the `domainName` parameter is used.

If neither host header or domain name parameter are provided, the domain name from the assigned origin is used or the host header from the incoming request if the forward to origin (FTO) policy includes the host. The host header can't include a colon (`:`) and can't be an IP address. The host header can be up to 253 characters.

sni (optional, for non-S3 custom origins)

The Server Name Indication (SNI) is an extension to the Transport Layer Security (TLS) protocol by which a client indicates which hostname it is attempting to connect to at the start of the TLS handshaking process. This value should match a common name on a TLS certificate on your origin server, otherwise your origin server might throw an error.

If this is not provided, the value from the `hostHeader` parameter is used. If the host header is not provided, the value from the `domainName` parameter is used.

If neither host header or domain name parameter are provided, the domain name from the assigned origin is used or the host header from the incoming request if the forward to origin (FTO) policy includes the host. The SNI can't include a colon (`:`) and can't be an IP address. The SNI can be up to 253 characters.

allowedCertificateNames (optional, for non-S3 custom origins)

You can include a list of valid certificate names to be used by CloudFront to validate the domain matching from your origin server TLS certificate during the TLS handshake with your origin server. This field expects an array of valid domain names and can include wildcard domains, such as `*.example.com`.

You can specify up to 20 allowed certificate names. Each certificate name can have up to 64 characters.

selectionCriteria (optional)

Select whether to use the `default` origin failover criteria or to use the `media-quality-score` based failover logic. Valid values are as follows:

- `default` uses the failover criteria, based on the status codes that are specified in the `failoverCriteria`. If you don't set `selectionCriteria` in the function, `default` will be used.
- `media-quality-score` is used when the media aware routing capability is being used.

failoverCriteria (required)

An array of status codes that, when returned from the primary origin, will trigger CloudFront to failover to the secondary origin. If you overwrite an existing origin group, this array will overwrite all failover status codes that are set in the origin group's original configuration.

When you use `media-quality-score` `selectionCriteria`, CloudFront will attempt to route requests based on the media quality score. If the selected origin returns an error code set in this array, CloudFront will failover to the other origin.

Example– Create request origin group

The following example creates an origin group for a request using the origin IDs. These origin IDs come from the origin group configuration for the distribution used to run this function.

Optionally, you can use `originOverrides` to override the origin group configurations for `sni`, `hostHeader`, and `allowedCertificateNames`.

```
import cf from 'cloudfront';

function handler(event) {
    cf.createRequestOriginGroup({
        "originIds": [
            {
                "originId": "origin-1",
                "originOverrides": {
                    "hostHeader": "hostHeader.example.com",
                    "sni": "sni.example.com",
                    "allowedCertificateNames": ["cert1.example.com",
 "cert2.example.com", "cert3.example.com"]
                }
            },
            {
                "originId": "origin-2",
                "originOverrides": {
                    "hostHeader": "hostHeader2.example.com",
                    "sni": "sni2.example.com",
                    "allowedCertificateNames": ["cert4.example.com",
 "cert5.example.com"]
                }
            }
        ],
        "failoverCriteria": {
            "statusCodes": [500]
        }
    });

    event.request.headers['x-hookx'] = { value: 'origin-overrides' };
```

```
    return event.request;
}
```

Helper methods for CloudFront SaaS Manager properties

Use the following helper functions for CloudFront SaaS Manager to retrieve values for your multi-tenant distributions in the function that you create. To use examples on this page, you must first create a CloudFront function by using JavaScript runtime 2.0. For more information, see JavaScript runtime 2.0 features for CloudFront Functions.

Topics

- Connection groups
- Distribution tenants

Connection groups

The connection group that is associated with your distribution tenants has a domain name.

To get this value, use the `endpoint` field of the `context` subobject of the event object.

Request

```
const value = event.context.endpoint;
```

Response

The response is a `string` that contains the connection group's domain name, such as d111111abcdef8.cloudfront.net. The `endpoint` field only appears when your function is invoked for a multi-tenant distribution with an associated connection group. For more information, see Context object.

Distribution tenants

CloudFront Functions has a module that provides access to specific distribution tenant values.

To use this module, include the following statement in the first line of your function code:

```
import cf from 'cloudfront';
```

You can use the following examples only in the `handler` function, either directly or through any nested-call function.

`distributionTenant.id` field

Use this field to get the value of distribution tenant ID.

Request

```
const value = cf.distributionTenant.id;
```

Response

The response is a `string` that contains the distribution tenant ID, such as `dt_1a2b3c4d5e6f7`.

Error handling

If your function is invoked for a standard distribution, specifying the `distributionTenant.id` field will return the `distributionTenant module is not available` type error. To handle this use case, you can add a `try` and `catch` block to your code.

`distributionTenant.parameters.get()` method

Use this method to return the value for the distribution tenant parameter names that you specified.

```
distributionTenant.parameters.get("key");
```

`key`: The distribution tenant parameter name that you want to fetch the value for.

Request

```
const value = distributionTenant.parameters.get("key");
```

Response

The response is a `string` that contains the value for the distribution tenant parameter. For example, if your key name is `TenantPath`, then the value for this parameter might be `tenant1`.

Error handling

You might receive the following errors:

- If your function is invoked for a standard distribution, the `distributionTenant.parameters.get()` method will return the `distributionTenant module is not available` type error.
- The `DistributionTenantParameterKeyNotFound` error is returned when the distribution tenant parameter that you specified doesn't exist.

To manage these use cases, you can add a `try` and `catch` block to your code.

Use async and await

CloudFront Functions JavaScript runtime functions 2.0 provide `async` and `await` syntax to handle `Promise` objects. Promises represent delayed results that can be accessed by using the `await` keyword in functions marked as `async`.

For more information about `Promise` objects, see Promise.

> **Note**
>
> You must use JavaScript runtime 2.0 for the following code samples.
> `await` can be used inside `async` functions only. `async` arguments and closures are not supported.

```
async function answer() {
    return 42;
}

// Note: async, await can be used only inside an async function. async arguments and
 closures are not supported.

async function handler(event) {
    // var answer_value = answer(); // returns Promise, not a 42 value
    let answer_value = await answer(); // resolves Promise, 42
    console.log("Answer"+answer_value);
    event.request.headers['answer'] = { value : ""+answer_value };
    return event.request;
}
```

The following example JavaScript code shows how to view promises with the `then` chain method. You can use `catch` to view errors.

> **Warning**
>
> Using promise combinators (for example, `Promise.all` and `Promise.any`) and promise chain methods (for example, `then` and `catch`) can require high function memory usage. If your function exceeds the maximum function memory quota, it will fail to execute. To avoid this error, we recommend that you use the `await` syntax instead of `promise` methods. You also can't use promise combinators to make more than one network call. A function can make only one network call per invocation. Calling `Promise.all()` with more than one network call throws an error. For more information, see Helper methods for network calls.

```
async function answer() {
    return 42;
}

async function squared_answer() {
   return answer().then(value => value * value)
}
// Note: async, await can be used only inside an async function. async arguments and
 closures are not supported.
async function handler(event) {
    // var answer_value = answer(); // returns Promise, not a 42 value
    let answer_value = await squared_answer(); // resolves Promise, 42
    console.log("Answer"+answer_value);
    event.request.headers['answer'] = { value : ""+answer_value };
    return event.request;
}
```

CWT support for CloudFront Functions

This section provides details on support for CBOR Web Tokens (CWT) in your CloudFront Functions, which enables secure token-based authentication and authorization at CloudFront Edge Locations. This support is provided as a module, accessible in your CloudFront Function.

To use this module, create a CloudFront Function using JavaScript runtime 2.0 and include the following statement in the first line of the function code:

```
import cf from 'cloudfront';
```

The methods associated with this module are accessible through (where * is a wildcard representing the different functions present in the module):

```
cf.cwt.*
```

For more information, see JavaScript runtime 2.0 features for CloudFront Functions.

Currently, the module only supports MAC0 structure with HS256 (HMAC-SHA256) algorithm with a limit of 1 KB for the maximum token size.

Token structure

This section covers the token structure that is expected by the CWT module. The module expects the token to be correctly tagged and identifiable (for example COSE MAC0). For the token structure, the module follows the standards set by CBOR Object Signing and Encryption (COSE) [RFC 8152].

```
( // CWT Tag (Tag value: 61) --- optional
    ( // COSE MAC0 Structure Tag (Tag value: 17) --- required
        [
            protectedHeaders,
            unprotectedHeaders,
            payload,
            tag,
        ]
    )
)
```

Example: CWT using the COSE MAC0 structure

```
61( // CWT tag
    17( // COSE_MAC0 tag
        [
            { // Protected Headers
                1: 5  // algorithm
            },
            { // Unprotected Headers
                4: h'53796d6d6574726963323536' // kid : Symmetric key id
            },
```

```
            { // Payload
                1: "https://iss.example.com", // iss
                2: "exampleUser", // sub
                3: "https://aud.example.com", // aud
                4: 1444064944, // exp
                5: 1443944944, // nbf
                6: 1443944944, // iat
            },
            h'ac243fa30d37386146b4e559fdc56d28840c453e7caf58126e819cbc3c299b3a' // tag

        ]
    )
)
```

The tag in this example is the HMAC-SHA256 of the token, computed with the example key `Symmetric256` from RFC 8392.

> **Note**
>
> The CWT tag is optional when generating tokens. However, the COSE structure tag is required.

validateToken() method

The function decodes and validates a CWT token using the specified key. If validation is successful, it returns the decoded CWT token. Otherwise, it throws an error.

> **Important**
>
> `validateToken()` checks only the structure of the token and its tag. It doesn't check any claims, such as `exp`, `nbf`, or `aud`, or Common Access Token (CAT) claims, such as `catu`. Your function must check the claims that it relies on.

To check a `catu` query restriction, compare it with the unparsed query string, which `request.rawQueryString()` returns. For more information, see `rawQueryString()` method.

Request

```
cf.cwt.validateToken(token, handlerContext)
```

Parameters

token (Required)

Encoded token for validation. This must be a JavaScript Buffer.

handlerContext (Required)

A JavaScript Object that stores context for the validateToken call. At present, only the key property is supported.

key (Required)

Secret key for message digest computation. Can be provided either as a string or JavaScript Buffer. The bytes of a string are used as the key, so if you store the key base64-encoded, decode it first, for example with `Buffer.from(key, 'base64')`.

Response

When the `validateToken()` method returns a successfully validated token, the response from the function is a JavaScript object in the following format. Once decoded, all claim keys are represented as strings of their numeric labels, such as `"4"` for `exp`.

```
{
    protected,
    unprotected,
    payload
}
```

Example - Validate token with kid sent in a request header

This example demonstrates CWT token validation, where the `kid` is read from the `x-cwt-kid` request header. The `kid` is then passed into CloudFront Functions KeyValueStore to fetch the secret key used to validate the token.

```
import cf from 'cloudfront'

const CwtClaims = {
   iss: 1,
   aud: 3,
   exp: 4
}
```

```
async function handler(event) {
    try {
        let request = event.request;
        let encodedToken = request.headers['x-cwt-token'].value;
        let kid = request.headers['x-cwt-kid'].value;

        // Retrieve the secret key from the kvs
        let secretKey = await cf.kvs().get(kid);

        // Now you can use the secretKey to decode & validate the token.
        let tokenBuffer = Buffer.from(encodedToken, 'base64url');

        let handlerContext = {
           key: secretKey,
        }

        try {
            let cwtObj = cf.cwt.validateToken(tokenBuffer, handlerContext);

            // Check if token is expired
            const currentTime = Math.floor(Date.now() / 1000); // Current time in
 seconds
            if (cwtObj.payload[CwtClaims.exp] && cwtObj.payload[CwtClaims.exp] <
 currentTime) {
                return {
                    statusCode: 401,
                    statusDescription: 'Token expired'
                };
            }
        } catch (error) {
            return {
               statusCode: 401,
               statusDescription: 'Invalid token'
            };
         }
    } catch (error) {
        return {
            statusCode: 500,
            statusDescription: 'Token processing failed'
        };
     }
    return request;
}
```

generateToken() method

This function generates a new CWT token using the provided payload and context settings.

Request

```
cf.cwt.generateToken(tokenContents, generatorContext)
```

Parameters

tokenContents (Required)

The contents of the token to encode, as a JavaScript object with exactly three properties: `protected`, `unprotected`, and `payload`. This is the same format as the object that `validateToken()` returns.

generatorContext (Required)

This is a JavaScript Object that is used as the context for generating the token and contains the following key value pairs:

cwtTag (Optional)

This value is a boolean, which if `true` specifies the `cwtTag` should be added.

coseTag (Required)

Specifies the COSE tag type. Currently only supports `MAC0`.

key (Required)

Secret key to compute message digest. This value can be either a string or JavaScript `Buffer`.

Response

Returns a JavaScript Buffer containing the encoded token.

Example: Generate a token

```
import cf from 'cloudfront';

const CwtClaims = {
    iss: 1,
    sub: 2,
```

```
    exp: 4
};

const CatClaims = {
    catnip: 311,
    catu: 312,
    catm: 313,
    catr: 323
};

const Catu = {
    host: 1,
    path: 3,
    ext: 8
};

const CatuMatchTypes = {
    exact_match: 0,
    prefix_match: 1,
    suffix_match: 2
};

const Catr = {
    type: 0,
    expadd: 1,
    deadline: 2,
    header_name: 4
};

const CatrTypes = {
    automatic: 0,
    cookie: 1,
    header: 2,
    redirect: 3
};

async function handler(event) {
    try {
        const response = {
            statusCode: 200,
            statusDescription: 'OK',
            headers: {}
        };
```

```
        const commonAccessToken = {
            protected: {
                1: 5,
            },
            unprotected: {},
            payload: {
                [CwtClaims.iss]: "cloudfront-documentation",
                [CwtClaims.sub]: "cwt-support-on-cloudfront-functions",
                [CwtClaims.exp]: 1740000000,
                [CatClaims.catu]: {
                    [Catu.host]: {
                        [CatuMatchTypes.suffix_match]: ".cloudfront.net"
                    },
                    [Catu.path]: {
                        [CatuMatchTypes.prefix_match]: "/media/live-stream/cf-4k/"
                    },
                    [Catu.ext]: {
                        [CatuMatchTypes.exact_match]: [
                            ".m3u8",
                            ".ts",
                            ".mpd"
                        ]
                    }
                },
                [CatClaims.catnip]: [
                    "[IP_ADDRESS]",
                    "[IP_ADDRESS]"
                ],
                [CatClaims.catm]: [
                    "GET",
                    "HEAD"
                ],
                [CatClaims.catr]: {
                    [Catr.type]: CatrTypes.header,
                    [Catr.expadd]: 3600,
                    [Catr.deadline]: 60,
                    [Catr.header_name]: "CTA-Common-Access-Token"
                }
            }
        };

        const request = event.request;
        if (!request.headers['x-cwt-kid']) {
            throw new Error('Missing x-cwt-kid header');
```

```
        }

        const kid = request.headers['x-cwt-kid'].value;
        const secretKey = await cf.kvs().get(kid);

        if (!secretKey) {
            throw new Error('Secret key not found for provided kid');
        }

        try {
            const genContext = {
                cwtTag: true,
                coseTag: "MAC0",
                key: secretKey
            };

            const tokenBuffer = cf.cwt.generateToken(commonAccessToken, genContext);
            response.headers['x-generated-cwt-token'] = { value:
 tokenBuffer.toString('base64url') };

            return response;
        } catch (tokenError) {
            return {
                statusCode: 500,
                statusDescription: 'Could not generate the token'
            };
        }
    } catch (error) {
        return {
            statusCode: 500,
            statusDescription: 'Token processing failed'
        };
    }
}
```

Example: Refresh token based on some logic

```
import cf from 'cloudfront'

const CwtClaims = {
   iss: 1,
   aud: 3,
   exp: 4
```

```
}

async function handler(event) {
    try {
        let request = event.request;
        let encodedToken = request.headers['x-cwt-token'].value;
        let kid = request.headers['x-cwt-kid'].value;
        let secretKey = await cf.kvs().get(kid); // Retrieve the secret key from the
 kvs

        // Now you can use the secretKey to decode & validate the token.
        let tokenBuffer = Buffer.from(encodedToken, 'base64url');

        let handlerContext = {
           key: secretKey,
        }

        try {
            let cwtJSON = cf.cwt.validateToken(tokenBuffer, handlerContext);

            // Check if token is expired
            const currentTime = Math.floor(Date.now() / 1000); // Current time in
 seconds
            if (cwtJSON.payload[CwtClaims.exp] && cwtJSON.payload[CwtClaims.exp] <
 currentTime) {
                // We can regenerate the token and add 8 hours to the expiry time
                cwtJSON.payload[CwtClaims.exp] = Math.floor(Date.now() / 1000) + (8 *
 60 * 60);

                let genContext = {
                  coseTag: "MAC0",
                  key: secretKey
                }

                let newTokenBuffer = cf.cwt.generateToken(cwtJSON, genContext);
                 request.headers['x-cwt-regenerated-token'] = { value:
 newTokenBuffer.toString('base64url') };
            }
        } catch (error) {
            return {
               statusCode: 401,
               statusDescription: 'Invalid token'
            };
         }
```

```
    }
    catch (error) {
        return {
            statusCode: 500,
            statusDescription: 'Token processing failed'
        };
     }
    return request;
}
```

General helper methods

This page provides additional helper methods inside CloudFront Functions. To use these methods, create a CloudFront function using JavaScript runtime 2.0.

```
import cf from 'cloudfront';
```

For more information, see JavaScript runtime 2.0 features for CloudFront Functions.

`edgeLocation` metadata

This method requires using the `cloudfront` module.

> **Note**
>
> You can only use this method for viewer-request functions. For viewer-response functions, this method is empty.

Use this JavaScript object to obtain the edge location airport code, expected Regional Edge Cache region or the CloudFront server IP address used to handle the request. This metadata is available only the viewer request event trigger.

```
cf.edgeLocation = {
    name: SEA
    serverIp: 1.2.3.4
    region: us-west-2
}
```

The `cf.edgeLocation` object can contain the following:

name

The three-letter IATA code of the edge location that handled the request.

serverIp

The IPv4 or IPv6 address of the server that handled the request.

region

The CloudFront Regional Edge Cache (REC) that the request is *expected* to use if there is a cache miss. This value is not updated in the event that the expected REC is unavailable and a backup REC is used for the request. This doesn't include the Origin Shield location being used, except in cases when the primary REC and the Origin Shield are the same location.

> **Note**
>
> CloudFront Functions isn't invoked a second time when CloudFront is configured to use origin failover. For more information, see Optimize high availability with CloudFront origin failover.

`rawQueryString()` method

This method doesn't require the `cloudfront` module.

Use the `rawQueryString()` method to retrieve the unparsed and unaltered query string as a string.

Request

```
function handler(event) {
    var request = event.request;
    const qs = request.rawQueryString();
}
```

Response

Returns the full query string of the incoming request as a string value without the leading ?.

- If there isn't a query string, but the ? is present, the function returns an empty string.

- If there isn't a query string and the ? isn't present, the function returns `undefined`.

Case 1: Full query string returned (without leading ?)

Incoming request URL: `https://example.com/page?name=John&age=25&city=Boston`

`rawQueryString()` returns: `"name=John&age=25&city=Boston"`

Case 2: Empty string returned (when ? is present but without parameters)

Incoming request URL: `https://example.com/page?`

`rawQueryString()` returns: `""`

Case 3: `undefined` returned (no query string and no ?)

Incoming request URL: `https://example.com/page`

`rawQueryString()` returns: `undefined`

`logCustomData()` method

To use this method, import the `cloudfront` module.

Use the `logCustomData()` method to send custom data from your Amazon CloudFront function into CloudFront access logs. The data is written to the `viewer-request-log-data` or `viewer-response-log-data` log field, depending on whether the function is associated with a viewer request or viewer response event.

The method accepts a single string argument. To log an object, use `JSON.stringify()` to convert it to a string first. For example:

```
const resp_obj = JSON.stringify(some_obj);
cf.logCustomData(resp_obj);
```

Each field supports up to 800 bytes of data. CloudFront automatically URL-encodes the data before writing it to the log field. CloudFront truncates any data above 800 bytes after URL-encoding. If your function calls `logCustomData()` multiple times within a single execution, CloudFront uses only the last value.

Add fields to your log configuration

To receive this data in your CloudFront access logs, you must add the `viewer-request-log-data` or `viewer-response-log-data` fields to your real-time log configuration or standard logging (v2) setup.

Difference from console.log()

The `logCustomData()` method does not replace `console.log()` for CloudFront Functions. Use `console.log()` to send log lines to Amazon CloudWatch Logs. Use `cf.logCustomData()` to write custom data to CloudFront access logs (real-time logs and standard logging (v2)). You can use both methods in the same function. For more information, see CloudFront Functions logs.

The method uses the following syntax:

```
cf.logCustomData(String);
```

Example Log a header in a viewer request function

```
import cf from 'cloudfront';

function handler(event) {
    var request = event.request;

    // Check if the debug header exists, if so log it
    if (request.headers['x-debug-header']) {
        // Log the debug header value
        cf.logCustomData("debug header found: " + request.headers['x-debug-
header'].value);
    }

    return request;
}
```

Helper methods for mutual TLS

CloudFront provides mTLS-specific helper functions in the `cf.mtls` namespace for viewer request functions. These functions rename, reformat, or combine client certificate headers before forwarding the request to your origin.

```
import cf from "cloudfront";
```

`cf.mtls.renameClientCertHeaders`

Renames certificate metadata headers to custom header names.

```
cf.mtls.renameClientCertHeaders({
    "CloudFront-Viewer-Cert-Pem": "X-Client-Cert",
    "CloudFront-Viewer-Cert-Serial-Number": "X-Client-Cert-Serial",
    "CloudFront-Viewer-Cert-Issuer": "X-Client-Cert-Issuer",
});
```

Allowed source headers:

- `CloudFront-Viewer-Cert-Pem`
- `CloudFront-Viewer-Cert-Serial-Number`
- `CloudFront-Viewer-Cert-Issuer`
- `CloudFront-Viewer-Cert-Subject`
- `CloudFront-Viewer-Cert-Validity`
- `CloudFront-Viewer-Cert-Present`
- `CloudFront-Viewer-Cert-Sha256`
- `Client-Cert`
- `Client-Cert-Chain`

`cf.mtls.renamePemHeaders`

Renames PEM certificate headers and optionally reformats the certificate encoding.

```
cf.mtls.renamePemHeaders({
    "Client-Cert": {
```

```
        "newHeaderName": "X-Leaf-Cert",
        "pemCertFormatInfo": {
            "certHeader": "-----CUSTOM HEADER-----",
            "certFooter": "-----CUSTOM FOOTER-----",
            "certEndMarker": "",
            "keepNewlinesInCertData": true
        }
    },
    "Client-Cert-Chain": {
        "newHeaderName": "X-Intermediate-Certs",
        "pemCertFormatInfo": {
            "certHeader": "-----CUSTOM HEADER-----",
            "certFooter": "-----CUSTOM FOOTER-----",
            "certEndMarker": "",
            "keepNewlinesInCertData": true
        }
    }
});
```

Allowed source headers:

- In passthrough mode: `Client-Cert`, `Client-Cert-Chain`
- In required/optional mode: `CloudFront-Viewer-Cert-PEM`

`pemCertFormatInfo` fields:

For `CloudFront-Viewer-Cert-PEM`:

- `certHeader` replaces `-----BEGIN CERTIFICATE-----`.
- `certFooter` replaces `-----END CERTIFICATE-----`.
- `certEndMarker` sets a custom string after the certFooter.
- `keepNewlinesInCertData` (default: `true`) preserves newlines in base64 data when true.

For `Client-Cert` and `Client-Cert-Chain`:

- `certHeader` replaces `:`.
- `certFooter` replaces `:`.
- `certEndMarker` sets a custom string after the certFooter.
- `keepNewlinesInCertData` (default: `false`) preserves newlines in base64 data when true.

`cf.mtls.combinePemHeaders`

Combines `Client-Cert` and `Client-Cert-Chain` into a single header containing the full certificate chain.

```
cf.mtls.combinePemHeaders({
    "newHeaderName": "X-Full-Chain",
    "pemCertFormatInfo": {
        "certHeader": "-----BEGIN CERTIFICATE-----",
        "certFooter": "-----END CERTIFICATE-----",
        "certEndMarker": "\n",
        "keepNewlinesInCertData": false
    }
});
```

`certEndMarker` sets the delimiter between certificates.

Note

- These helper functions can be used in all mTLS modes (required, optional, and passthrough). However, `cf.mtls.combinePemHeaders` only has effect in passthrough mode — in required and optional modes, the `Client-Cert` and `Client-Cert-Chain` headers are not present, so the function is a no-op.
- If `cf.mtls.*` methods and `cf.updateRequestOrigin()` with `customHeaders` target the same header name, CloudFront returns a 502 error.
- In Passthrough mode, CloudFront drops any incoming `Client-Cert` or `Client-Cert-Chain` headers and re-adds them from the actual client certificate.
- CloudFront doesn't present the full raw PEM certificate contents in any edge functions.

Helper methods for network calls

This section applies if you send an HTTP request to an origin from inside your CloudFront Functions code. You can make network calls on *viewer request* or *viewer response* CloudFront Functions. CloudFront Functions has a module that provides helper methods to make network calls.

To use this module, create a CloudFront function using JavaScript runtime 2.0 and include the following statement in the first line of the function code:

```
import cf from "cloudfront";
```

For more information, see JavaScript runtime 2.0 features for CloudFront Functions.

The destination of the network call must be an origin that is already configured on the same distribution. You can't pass a URL with a host name. You must pass a path and the ID of the origin. In the CloudFront console this is the **Name** field on the origin, which is the same value that the CloudFront API and AWS CloudFormation call the origin `Id`. For more information, see Origin settings.

Sending the call through an origin takes advantage of CloudFront's origin handling capabilities. You don't have to reimplement any of the following in your function code:

- Origin access control (OAC) signs the request with SigV4. You can call a private Amazon S3 bucket, a Lambda function URL, or another AWS origin without putting credentials in your function code.
- With VPC origins, you can call a service in a private subnet, so the destination never needs a public endpoint.
- Connection reuse means CloudFront usually already holds a warm connection to the origin, so the call avoids TCP and TLS setup.
- Custom origin headers configured on the origin are sent on the call, so shared secrets and routing headers stay in your distribution configuration rather than in function code.
- Origin mutual TLS (mTLS) configured on the origin applies to the call. CloudFront presents the origin's client certificate, so your function doesn't handle certificates or private keys.

The response timeout is the one origin setting that does not carry over. A network call times out after 1 second even when the origin is configured with a longer response timeout.

> **Note**
>
> Distributions on the free pricing plan don't support network calls. For more information, see CloudFront flat-rate pricing plans.

Note

Network calls are not supported when you test a function with the `TestFunction` API or the **Test** tab in the CloudFront console. In a test, `cf.fetch()` and `cf.beacon()` throw a `TypeError` (`"cloudfront.fetch" is not available in test invocations`). If your function does not catch the error, the test fails. Wrap the call in `try` and `catch` so the rest of the function can be tested.

Important

A function can make one network call per invocation. Both `fetch()` and `beacon()` count toward this one network call. A second network call throws an error.

Topics

- Choose between a network call and CloudFront KeyValueStore
- fetch() method
- beacon() method
- How CloudFront builds the request
- Limitations

Choose between a network call and CloudFront KeyValueStore

You can read data inside a function by using either a network call or Amazon CloudFront KeyValueStore. Your choice depends on how often the data changes and how current the answer needs to be.

A network call is most useful in the following situations:

- When the data changes between requests and you need a current answer at request time, such as whether a token has been revoked or which segment a viewer belongs to.
- When the dataset is too large to fit in a key value store. A key value store has a maximum total size, so a large dataset such as a full user table or a product catalog has to stay in the service that holds it.

CloudFront KeyValueStore is most useful in the following situations:

- When the data is static or changes infrequently, such as redirect maps, lookup tables, or feature flags that change daily. CloudFront KeyValueStore reads at the edge with no external call, which gives better performance at lower cost.
- When you need more than one lookup for a single request. A function can make only one network call per invocation, but it can read from CloudFront KeyValueStore repeatedly.

A function can use both. You might read an experiment configuration from CloudFront KeyValueStore to decide which test is active, then make a network call to assign the viewer a variant.

For more information about CloudFront KeyValueStore, see Helper methods for key value stores.

`fetch()` method

Use the `fetch()` method to send a request to an origin and read the response. The method returns a promise, so it must be used with `await` inside an `async` function.

Request

```
cf.fetch(path, options)
```

> **Important**
>
> Option names are case sensitive, and unknown options are ignored. For example, `Method: 'POST'` is ignored and the request is sent as `GET`, as with standard `fetch()`. If the call also has a body, it throws a `TypeError`, because a body is not allowed with `GET`.

The parameters can contain the following:

`path` (required)

The URI path, and optionally a query string, of the request. For example, `/v1/authorize?scope=read`. The path must start with `/`. CloudFront sends the request to the origin named in `options.originId` and appends the path to the origin's domain name. If the origin has an origin path configured, the path is appended after the origin path.

`options` (required)

The options can contain the following:

`originId` (required)

The ID of the origin to send the request to. Must exactly match the ID of an origin on the same distribution. In the CloudFront console, this is the **Name** field on the origin.

`method` (optional)

The HTTP method. One of `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, `DELETE`, or `OPTIONS`. If this is not provided, `GET` is used.

The value must be upper case. Unlike standard `fetch()`, the method is not normalized, so a value such as `'post'` throws a `TypeError`.

`headers` (optional)

Request headers as a plain object. Each property name is a header name and each value is a string, for example `{ 'content-type': 'application/json' }`.

This format is not the same as `event.request.headers`. In the event structure, headers use the shape `{ 'content-type': { value: 'application/json' } }`. Copying the event shape into the options throws an error.

For the headers that you can't set, see How CloudFront builds the request.

`body` (optional)

The request body. Can be a string, an `ArrayBuffer`, or an `ArrayBufferView` such as a `Uint8Array`. To send JSON, use `JSON.stringify()`.

A body is not allowed with `GET` or `HEAD`, including when `method` is not provided and the default `GET` is used. A call with a body and one of these methods throws a `TypeError` (`options.body is not allowed with GET or HEAD`).

`includeBody` (optional)

Whether CloudFront reads the response body and makes it available to your function. If this is not provided, `false` is used. Set it to `true` when you need the body.

Without `includeBody: true`, every body accessor reads empty: `text()` resolves to `""`, `json()` resolves to `null`, and `bytes()` and `arrayBuffer()` resolve to empty values. Nothing is consumed, so `bodyUsed` stays `false`.

Response

The method returns a promise that resolves to a `Response` object. The object can contain the following:

`status`

The HTTP status code from the origin as a number.

ok

`true` if `status` is 200 to 299, otherwise `false`.

`headers`

A read-only property that holds the response headers as a Headers object. It supports the `get()`, `has()`, `getSetCookie()`, `forEach()`, `keys()`, `values()`, and `entries()` methods, and you can iterate over it.

Header names are case insensitive in `get()` and `has()`. `get()` returns `null` when the header is not present. `getSetCookie()` returns an array of every `Set-Cookie` value.

```
const type = response.headers.get('content-type');
const cookies = response.headers.getSetCookie();
```

`text()`

A promise that resolves to the body decoded as UTF-8 text. Invalid byte sequences are replaced.

`json()`

A promise that resolves to the body parsed as JSON. Rejects with `SyntaxError` if the body is not valid JSON.

`bytes()`

A promise that resolves to the raw body bytes as a `Uint8Array`. Buffer methods such as `toString('base64')` work on the result.

`arrayBuffer()`

A promise that resolves to the raw body bytes as an `ArrayBuffer`.

`bodyUsed`

`true` once the body has been read.

> **Important**
>
> You can read the body only once. After `text()`, `json()`, `bytes()`, or `arrayBuffer()`, any further read rejects with `TypeError: body already read`. Use bodyUsed to check whether it has been read. If you need the body more than once, keep the first result in a variable.

A network call response carries its own `x-amz-cf-id`, and it is not the same as the one on the viewer request that triggered your function. They are two separate requests with two separate identifiers. If you are investigating the network call, the identifier from the network call response is the one that matters. You can read it with `response.headers.get('x-amz-cf-id')`.

Error handling

Wrap every `await cf.fetch()` in a `try` and `catch` block. Then decide whether your function should fail open and let the request through, or fail closed and block it. If an error is not caught, the function fails and CloudFront returns HTTP 503 to the viewer. This is the same as any other unhandled function error.

Invalid arguments reject the promise that `fetch()` returns, so an `await` inside a `try` and `catch` block handles both argument errors and errors from the call itself.

The method can throw the following errors:

`TypeError`

The call is invalid or the origin couldn't be reached. This could be from a bad argument, a disallowed header, a request or response that is too large, an unknown origin ID, or a distribution that does not support network calls.

It also covers an origin that CloudFront could not reach or complete the call with: a DNS failure, a connection failure, a TLS failure, origin response headers that are too large for CloudFront, or the origin closing the connection. The message names the reason, for example `The request could not be completed due to OriginDnsError`. Origins that resolve to loopback or reserved IP addresses are refused in the same way, with `OriginDnsError`.

`TimeoutError`

The origin did not return a complete response within 1 second. The limit applies to the whole call, including connection setup and reading the body.

`InternalError`

The call failed inside CloudFront.

`SyntaxError`

Thrown by `response.json()` when the body is not valid JSON. Check what your origin returns.

> **Note**
>
> A TLS failure to a distant origin, or an origin on a private IP address, can surface as a `TimeoutError` rather than an origin error, because the 1 second limit is reached first. To call a service in a private network, use VPC origins.

> **Warning**
>
> Only one network call is allowed per invocation, so promise combinators can't be used to make more than one. Passing two `cf.fetch()` calls to `Promise.all()` throws an error. For more information about promise combinators and function memory, see Use async and await.

Example: Authorization check that fails closed

This function only lets the request through if the authorization service approves it. If the service can't be reached, the request is blocked.

Replace `/v1/authorize` with the path on your origin, and `auth-service` with the ID of your origin.

The following JavaScript example sends the viewer's token to an authorization service:

```
import cf from 'cloudfront';

async function handler(event) {
    const request = event.request;
    const auth = request.headers['authorization'];
```

```
    if (!auth) {
        return { statusCode: 401, statusDescription: 'Unauthorized' };
    }

    try {
        const response = await cf.fetch('/v1/authorize', {
            originId: 'auth-service',
            method: 'POST',
            headers: { 'content-type': 'application/json' },
            body: JSON.stringify({ token: auth.value, uri: request.uri }),
            includeBody: true,
        });

        if (!response.ok) {
            console.log('authorize returned ' + response.status);
            return { statusCode: 403, statusDescription: 'Forbidden' };
        }

        const result = await response.json();
        request.headers['x-user-id'] = { value: result.userId };
        return request;
    } catch (e) {
        console.log('Authorization check failed: ' + e.name + ': ' + e.message);
        return { statusCode: 503, statusDescription: 'Service Unavailable' };
    }
}
```

Example: Read dynamic configuration from Amazon S3

This function reads a configuration object from an Amazon S3 origin and uses it to tag the request. The `method` option is omitted because `GET` is the default. If the Amazon S3 bucket is private and origin access control (OAC) is set up on the origin, CloudFront signs the request to the bucket. This function fails open, because a missing configuration should not block the viewer.

Replace `/config/experiments.json` with the path of your object, and `config-bucket` with the ID of your origin.

The following JavaScript example reads the configuration object:

```
import cf from 'cloudfront';

async function handler(event) {
```

```
    const request = event.request;

    try {
        const response = await cf.fetch('/config/experiments.json', {
            originId: 'config-bucket',
            includeBody: true,
        });

        if (response.ok) {
            const config = await response.json();
            request.headers['x-experiment'] = { value: config.activeExperiment };
        }
    } catch (e) {
        console.log('Config lookup failed: ' + e.name);
    }

    return request;
}
```

beacon( ) method

Use the `beacon( )` method to send a request to an origin without waiting for the response. Use it for analytics events, audit records, and anything else where the viewer should not wait on delivery. A beacon counts against the network call quota. Each invocation can make one network call in total, using either `fetch( )` or `beacon( )`, not one of each. Call `beacon( )` inside the handler. It doesn't work from global scope.

> **Important**
>
> A beacon is best effort. Don't use it for anything that must succeed. Argument errors throw immediately and can be caught in a `try` and `catch` block, but failures that happen after the function has moved on are not surfaced to your function. Setting `logDeliveryErrors` to `true` is the only way to see those errors.

Request

```
cf.beacon(path, options)
```

The parameters can contain the following:

`path` (required)

The URI path, and optionally a query string, of the request. For example, `/v1/events?source=edge`. The path must start with `/`. CloudFront sends the request to the origin named in `options.originId` and appends the path to the origin's domain name. If the origin has an origin path configured, the path is appended after the origin path.

`options` (required)

The options can contain the following:

`originId` (required)

The ID of the origin to send the request to. Must exactly match the ID of an origin on the same distribution. In the CloudFront console, this is the **Name** field on the origin.

`method` (optional)

The HTTP method. One of `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, `DELETE`, or `OPTIONS`. Must be upper case. If this is not provided, `GET` is used.

`headers` (optional)

Request headers as a plain object. Each property name is a header name and each value is a string, for example `{ 'content-type': 'application/json' }`.

This format is not the same as `event.request.headers`. In the event structure, headers use the shape `{ 'content-type': { value: 'application/json' } }`. Copying the event shape into the options throws an error.

`body` (optional)

The request body. Can be a string, an `ArrayBuffer`, or an `ArrayBufferView` such as a `Uint8Array`. To send JSON, use `JSON.stringify()`. A body is not allowed with `GET` or `HEAD`, including the default `GET`, and throws a `TypeError`.

`logDeliveryErrors` (optional)

Whether delivery failures are written to CloudWatch Logs. If this is not provided, `false` is used. Set it to `true` to see beacons that fail after your function has returned. The errors go to the same log group as `console.log()` messages from the function. For more information, see CloudFront Functions logs.

A beacon counts as delivered when the origin returns a 2xx or 3xx status. Any other origin status is logged as `"cloudfront.beacon": origin returned status`.

> **Note**
>
> The `includeBody` option of the `fetch()` method is not supported, because there is no response to read. Passing this option throws a `TypeError`.

Response

The method returns `undefined` immediately and your function continues.

Example: Send an analytics event

The handler is not `async`, because there is nothing to await.

The following JavaScript example sends an event without waiting for a response:

```
import cf from 'cloudfront';

function handler(event) {
    const request = event.request;

    cf.beacon('/events', {
        originId: 'analytics-service',
        method: 'POST',
        headers: { 'content-type': 'application/json' },
        body: JSON.stringify({ uri: request.uri, viewerIp: event.viewer.ip }),
        logDeliveryErrors: true,
    });

    return request;
}
```

How CloudFront builds the request

CloudFront constructs each network call from scratch. Except for the proxy headers listed below, it does not copy the viewer's headers, cookies, or query string onto the request. If your origin needs something from the viewer request, pass it explicitly in `headers` or `body`.

You can set the `host` header on a network call to a custom origin that doesn't use origin access control. This differs from the viewer request, where the `host` header in the Request object is read only. Use it when the destination routes by host, such as a service that serves multiple hostnames from one endpoint. On an Amazon S3 origin or an origin with origin access control, setting `host`

throws a `TypeError` (`Host header override is not supported for the provided origin`). An invalid `host` value also throws a `TypeError`.

If the origin has origin access control (OAC) configured, CloudFront signs the network call with SigV4 on your behalf, so you don't implement signing in your function or put credentials in your function code. Signing follows the signing behavior of the origin. With `never`, CloudFront doesn't sign the call. With `no-override`, CloudFront doesn't sign the call when your function sets an `Authorization` header or the origin has an `Authorization` custom header. For more information, see Advanced settings for origin access control.

CloudFront adds the `x-amz-content-sha256` header for you. It is `UNSIGNED-PAYLOAD` for a call with a body and the hash of an empty string for a call without one. The exception is a `POST` or `PUT` with a body to a Lambda function URL, which doesn't accept unsigned payloads. For those calls, your function must add an `x-amz-content-sha256` header with the SHA-256 digest of the body.

CloudFront copies the following headers from the viewer request. If you supply them in your function, CloudFront ignores them.

- `X-Forwarded-For`
- `Via`
- `CDN-Loop`

In addition to the existing Disallowed headers for CloudFront Functions, CloudFront doesn't allow the following headers on network calls. Supplying any of them throws a `TypeError`.

- `Accept-Encoding`
- `Content-Length`
- `TE`
- `Transfer-Encoding`

Limitations

- Distributions on the free pricing plan do not support network calls. `cf.fetch()` and `cf.beacon()` throw a `TypeError`, for example `"cloudfront.fetch" is not available on the free pricing plan`. For more information, see CloudFront flat-rate pricing plans.
- Multi-tenant distributions and distribution tenants do not support network calls. For more information, see Unsupported features.

- Call `cf.fetch()` and `cf.beacon()` only inside the handler. They don't work from global scope.
- Only one network call per invocation.
- Network calls are not supported when you test a function with the `TestFunction` API or the **Test** tab in the CloudFront console. The call throws a `TypeError`.
- CloudFront Functions doesn't support response body streaming. `text()` decodes the body as UTF-8 and replaces invalid bytes. `bytes()` and `arrayBuffer()` return the raw bytes.
- AWS WAF does not inspect network calls. The web ACL associated with your distribution evaluates viewer requests, not requests your function makes.
- CloudFront doesn't cache network call responses. Every invocation that reaches a network call makes one.
- Connection Functions do not support network calls.
- Distributions that use continuous deployment do not support network calls.

For quotas, including the network call timeout and the maximum request and response body sizes, see Quotas on CloudFront Functions network calls.

Create functions

You create a function in two stages:

1. Create the function code as JavaScript. You can use the default example from the CloudFront console or write your own. For more information, see the following topics:
   - Write function code
   - the section called “Event structure”
   - CloudFront Functions examples for CloudFront
2. Use CloudFront to create the function and include your code. The code exists inside the function (not as a reference).

Console

To create a function

1. Sign in to the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions and choose the **Functions** page.
2. Choose **Create function**.

3. Enter a function name that is unique within the AWS account, choose the JavaScript version, and then choose **Continue**. The details page for the new function appears.

   > **Note**
   >
   > To use key-value pairs in the function, you must choose JavaScript runtime 2.0.
   > If you encounter a `TaggingConflict` (HTTP 409) error when creating a function with tags, wait a few minutes and retry, or create the function without tags initially and add them afterward.

4. In the **Function code** section, choose the **Build** tab and enter your function code. The sample code that is included in the **Build** tab illustrates the basic syntax for the function code.
5. Choose **Save changes**.
6. If the function code uses key-value pairs, you must associate a key value store.

   You can associate the key value store when you first create the function. Or, you can associate it later, by updating the function.

   To associate a key value store now, follow these steps:

   - Go to the **Associate KeyValueStore** section and choose **Associate existing KeyValueStore**.
   - Select the key value store that contains the key-value pairs in the function, and then choose **Associate KeyValueStore**.

   CloudFront immediately associates the store with the function. You don't need to save the function.

CLI

If you use the CLI, you typically first create the function code in a file, and then create the function with the AWS CLI.

To create a function

1. Create the function code in a file, and store it in a directory that your computer can connect to.

2. Run the command as shown in the example. This example uses the `fileb://` notation to pass in the file. It also includes line breaks to make the command more readable.

```
aws cloudfront create-function \
    --name MaxAge \
    --function-config '{"Comment":"Max Age 2 years","Runtime":"cloudfront-
js-2.0","KeyValueStoreAssociations":{"Quantity":1,"Items":
[{"KeyValueStoreARN":"arn:aws:cloudfront::111122223333:key-value-store/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"}]}}' \
    --function-code fileb://function-max-age-v1.js
```

Notes

- `Runtime` – The version of JavaScript. To use key value pairs in the function, you must specify version 2.0.
- `KeyValueStoreAssociations` – If your function uses key-value pairs, you can associate the key value store when you first create the function. Or, you can associate it later, by using `update-function`. The `Quantity` is always 1 because each function can have only one key value store associated with it.

When the command is successful, you see output like the following.

```
ETag: ETVABCEXAMPLE
FunctionSummary:
  FunctionConfig:
    Comment: Max Age 2 years
    Runtime: cloudfront-js-2.0
    KeyValueStoreAssociations= \
      {Quantity=1, \
      Items=[{KeyValueStoreARN='arn:aws:cloudfront::111122223333:key-value-
store/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111'}]} \
  FunctionMetadata:
    CreatedTime: '2021-04-18T20:38:56.915000+00:00'
    FunctionARN: arn:aws:cloudfront::111122223333:function/MaxAge
    LastModifiedTime: '2023-11-19T20:38:56.915000+00:00'
    Stage: DEVELOPMENT
  Name: MaxAge
  Status: UNPUBLISHED
```

```
Location: https://cloudfront.amazonaws.com/2020-05-31/function/
arn:aws:cloudfront:::function/MaxAge
```

Most of the information is repeated from the request. Other information is added by CloudFront.

Notes

- `ETag` – This value changes each time you modify the key value store. You use this value and the function name to reference the function in the future. Make sure that you always use the current `ETag`.
- `FunctionARN` – The ARN for your CloudFront function.
- `111122223333` – The AWS account.
- `Stage` – The stage of the function (`LIVE` or `DEVELOPMENT`).
- `Status` – The status of the function (`PUBLISHED` or `UNPUBLISHED`).

After you create the function, it's added to the `DEVELOPMENT` stage. We recommend that you test your function before you publish it. After you publish your function, the function changes to the `LIVE` stage.

TaggingConflict error when creating functions

CloudFront Functions reuse ARNs for functions with the same name. You may encounter a `TaggingConflict` (HTTP 409) error when creating a function with tags. This can occur in the following scenarios:

- **After deleting a function** – Recreating a function with the same name and tags immediately after deletion.
- **After a failed creation attempt** – If a previous create request failed (for example, due to invalid parameters, invalid KVS association, or a failed CloudFormation stack rollback).

If you receive a `TaggingConflict` error, you can:

- Wait a few minutes and retry the operation.
- Use a different function name. This avoids the ARN reuse conflict entirely.

- If your template creates both a KeyValueStore and a Function that references it, add an explicit `DependsOn` to make sure the KVS is fully provisioned before the function is created.

Test functions

Before you deploy the function to the live stage (production), you can test your function to verify that it works as intended. To test a function, you specify an *event object* that represents an HTTP request or response that your CloudFront distribution could receive in production.

CloudFront Functions does the following:

1. Runs the function, using the provided event object as input.
2. Returns the function's result (the modified event object) along with any function logs or error messages and the function's *compute utilization*. For more information about compute utilization, see the section called “Understand compute utilization”.

> **Note**
>
> When you test a function, CloudFront only validates against function execution errors. CloudFront doesn't validate whether the request will flow through successfully once published. For example, if your function deletes a required header, the test will succeed because there isn't an issue with the code. However, if you publish the function and associate it with a distribution, the function will fail when a request was made through CloudFront.

Contents

- Set up the event object
- Test the function
- Understand compute utilization

Set up the event object

Before you test a function, you must set up the event object to test it with. There are several options.

Option 1: Set up an event object without saving it

You can set up an event object in the visual editor in the CloudFront console and not save it.

You can use this event object to test the function from the CloudFront console, even though it's not saved.

Option 2: Create an event object in the visual editor

You can set up an event object in the visual editor in the CloudFront console and not save it. You can create 10 event objects for each function so that you can, for example, test different possible inputs.

When you create the event object in this way, you can use the event object to test the function in the CloudFront console. You can't use it to test the function using an AWS API or SDK.

Option 3: Create an event object using a text editor

You can use a text editor to create an event object in JSON format. For information about the structure of an event object, see Event structure.

You can use this event object to test the function using the CLI. But you can't use it to test the function in the CloudFront console.

To create an event object (option 1 or 2)

1. Sign in to the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions and choose the **Functions** page.

   Choose the function that you want to test.
2. On the function details page, choose the **Test** tab.
3. For **Event type**, choose one of the following options:

   - Choose **Viewer request** if the function modifies an HTTP request or generates a response based on the request. The **Request** section appears.

- Choose **Viewer response**. The **Request** and the **Response** sections appear.

4. Complete the fields to include in the event. You can choose **Edit JSON** to view the raw JSON.
5. (Optional) To save the event, choose **Save** and in the **Save test event**, enter a name and then choose **Save**.

   You can also choose **Edit JSON** and copy the raw JSON, and save it in your own file, outside of CloudFront.

To create an event object (option 3)

Create the event object using a text editor. Store the file in a directory that your computer can connect to.

Verify that you follow these guidelines:

- Omit the `distributionDomainName`, `distributionId`, and `requestId` fields.
- The names of headers, cookies, and query strings must be lowercase.

One option for creating an event object in this way is to create a sample using the visual editor. You can be sure that the sample is correctly formatted. You can then copy the raw JSON and paste it into a text editor and save the file.

For more information about the structure of an event, see Event structure.

Test the function

You can test a function in the CloudFront console or with the AWS Command Line Interface (AWS CLI).

Console

To test the function

1. Sign in to the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions and choose the **Functions** page.
2. Choose the function that you want to test.
3. Choose the **Test** tab.

4. Make sure that the correct event is displayed. To switch from the currently displayed event, choose another event in the **Select test event** field.
5. Choose **Test function**. The console shows the output of the function, including function logs and compute utilization.

CLI

You can test a function by using the **aws cloudfront test-function** command.

To test the function

1. Open a command line window.
2. Run the following command from the same directory that contains the specified file.

   This example uses the `fileb://` notation to pass in the event object file. It also includes line breaks to make the command more readable.

```
aws cloudfront test-function \
    --name MaxAge \
    --if-match ETVABCEXAMPLE \
    --event-object fileb://event-maxage-test01.json \
    --stage DEVELOPMENT
```

   Notes

   - You reference the function by its name and ETag (in the `if-match` parameter). You reference the event object by its location in your file system.
   - The stage can be `DEVELOPMENT` or `LIVE`.

   When the command is successful, you see output like the following.

```
TestResult:
  ComputeUtilization: '21'
  FunctionErrorMessage: ''
  FunctionExecutionLogs: []
  FunctionOutput: '{"response":{"headers":{"cloudfront-functions":
{"value":"generated-by-CloudFront-Functions"},"location":{"value":"https://
```

```
aws.amazon.com/cloudfront/"}},"statusDescription":"Found","cookies":
{},"statusCode":302}}'
  FunctionSummary:
    FunctionConfig:
      Comment: MaxAge function
      Runtime: cloudfront-js-2.0
      KeyValueStoreAssociations= \
      {Quantity=1, \
      Items=[{KeyValueStoreARN='arn:aws:cloudfront::111122223333:key-value-
store/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111'}]} \
    FunctionMetadata:
      CreatedTime: '2021-04-18T20:38:56.915000+00:00'
      FunctionARN: arn:aws:cloudfront::111122223333:function/MaxAge
      LastModifiedTime: '2023-17-20T10:38:57.057000+00:00'
      Stage: DEVELOPMENT
    Name: MaxAge
    Status: UNPUBLISHED
```

Notes

- `FunctionExecutionLogs` contains a list of log lines that the function wrote in `console.log()` statements (if any).
- `ComputeUtilization` contains information about running your function. See the section called “Understand compute utilization”.
- `FunctionOutput` contains the event object that the function returned.

Understand compute utilization

Compute utilization is the amount of time that the function took to run as a percentage of the maximum allowed time. For example, a value of 35 means that the function completed in 35% of the maximum allowed time.

If a function continuously exceeds the maximum allowed time, CloudFront throttles the function. The following list explains the likelihood of a function getting throttled based on the compute utilization value.

Compute utilization value:

- **1 – 50** – The function is comfortably below the maximum allowed time and should run without throttling.
- **51 – 70** – The function is nearing the maximum allowed time. Consider optimizing the function code.
- **71 – 100** – The function is very close to or exceeds the maximum allowed time. CloudFront is likely to throttle this function if you associate it with a distribution.

Update functions

You can update a function at any time. The changes are made only to the version of the function that is in the `DEVELOPMENT` stage. To copy the updates from the `DEVELOPMENT` stage to `LIVE`, you must publish the function.

You can update a function's code in the CloudFront console or with the AWS Command Line Interface (AWS CLI).

Console

To update function code

1. Sign in to the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions and choose the **Functions** page.

   Choose the function to update.
2. Choose **Edit** and make the following changes:

   - Update any fields in the **Details** section.
   - Change or remove the associated key value store. For more information about key value stores, see the section called " CloudFront KeyValueStore".
   - Change the function code. Choose the **Build** tab, make changes, then choose **Save changes** to save changes to the code.

CLI

To update the function code

1. Open a command line window.
2. Run the following command.

   This example uses the `fileb://` notation to pass in the file. It also includes line breaks to make the command more readable.

```
aws cloudfront update-function \
    --name MaxAge \
    --function-config '{"Comment":"Max Age 2 years","Runtime":"cloudfront-
js-2.0","KeyValueStoreAssociations":{"Quantity":1,"Items":
[{"KeyValueStoreARN":"arn:aws:cloudfront::111122223333:key-value-store/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"}]}}' \
    --function-code fileb://function-max-age-v1.js \
    --if-match ETVABCEXAMPLE
```

   Notes

   - You can identify the function by both its name and ETag (in the `if-match` parameter). Make sure that you use the current ETag. You can get this value from the DescribeFunction API operation.
   - You must include the `function-code`, even if you don't want to change it.
   - Be careful with the `function-config`. You should pass everything that you want to keep in the configuration. Specifically, handle the key value store as follows:
     - To retain the existing key value store association (if there is one), specify the name of the *existing* store.
     - To change the association, specify the name of the *new* key value store.
     - To remove the association, omit the `KeyValueStoreAssociations` parameter.

   When the command is successful, you see output like the following.

```
ETag: ETVXYZEXAMPLE
FunctionSummary:
  FunctionConfig:
    Comment: Max Age 2 years \
    Runtime: cloudfront-js-2.0 \
    KeyValueStoreAssociations= \
      {Quantity=1, \
      Items=[{KeyValueStoreARN='arn:aws:cloudfront::111122223333:key-value-
store/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111'}]} \
  FunctionMetadata: \
    CreatedTime: '2021-04-18T20:38:56.915000+00:00' \
    FunctionARN: arn:aws:cloudfront::111122223333:function/MaxAge \
    LastModifiedTime: '2023-12-19T23:41:15.389000+00:00' \
    Stage: DEVELOPMENT \
  Name: MaxAge \
  Status: UNPUBLISHED
```

Most of the information is repeated from the request. Other information is added by CloudFront.

Notes

- `ETag` – This value changes each time you modify the key value store.
- `FunctionARN` – The ARN for your CloudFront function.
- `Stage` – The stage for the function (`LIVE` or `DEVELOPMENT`).
- `Status` – The status of the function (`PUBLISHED` or `UNPUBLISHED`).

Publish functions

When you publish your function, this copies the function from the `DEVELOPMENT` stage to the `LIVE` stage.

If cache behaviors aren't associated with the function, publishing it enables you to associate it with a cache behavior. You can only associate cache behaviors with functions that are in the `LIVE` stage.

Important

- Before you publish, we recommend that you test the function.
- After you publish the function, all cache behaviors that are associated with that function automatically start using the newly published copy, as soon as the distributions finish deploying.

You can publish a function in the CloudFront console or with the AWS CLI.

Console

To publish a function

1. Sign in to the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions and choose the **Functions** page.
2. Choose the function to update.
3. Choose the **Publish** tab and then choose **Publish**. If your function is already attached to one or more cache behaviors, choose **Publish and update**.
4. (Optional) To see the distributions that are associated with the function, choose **Associated CloudFront distributions** to expand that section.

When successful, a banner appears at the top of the page that says ***Function name*** **published successfully**. You can also choose the **Build** tab and then choose **Live** to see the live version of the function code.

CLI

To publish a function

1. Open a command line window.
2. Run the following **aws cloudfront publish-function** command. In the example, line breaks are provided to make the example more readable.

```
aws cloudfront publish-function \
    --name MaxAge \
    --if-match ETVXYZEXAMPLE
```

When the command is successful, you see output like the following.

```
FunctionSummary:
  FunctionConfig:
    Comment: Max Age 2 years
    Runtime: cloudfront-js-2.0
  FunctionMetadata:
    CreatedTime: '2021-04-18T21:24:21.314000+00:00'
    FunctionARN: arn:aws:cloudfront::111122223333:function/ExampleFunction
    LastModifiedTime: '2023-12-19T23:41:15.389000+00:00'
    Stage: LIVE
  Name: MaxAge
  Status: UNASSOCIATED
```

Associate functions with distributions

To use a function with a distribution, you associate the function with one or more cache behaviors in the distribution. You can associate a function with multiple cache behaviors in multiple distributions.

You can associate a function with any of the following:

- An existing cache behavior
- A new cache behavior in an existing distribution
- A new cache behavior in a new distribution

When you associate a function with a cache behavior, you must choose an *event type*. The event type determines when CloudFront runs the function.

You can choose the following event types:

- **Viewer request** – The function runs when CloudFront receives a request from a viewer.
- **Viewer response** – The function runs before CloudFront returns a response to the viewer.

You can't use origin-facing event types (*origin request* and *origin response*) with CloudFront Functions. Instead you can use Lambda@Edge. For more information, see CloudFront events that can trigger a Lambda@Edge function.

> **Note**
>
> Before you associate a function, you must publish it to the LIVE stage.

You can associate a function with a distribution in the CloudFront console or with the AWS Command Line Interface (AWS CLI). The following procedure shows how to associate a function with an existing cache behavior.

Console

To associate a function with an existing cache behavior

1. Sign in to the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions and choose the **Functions** page.
2. Choose the function that you want to associate.
3. On the **Function** page, choose the **Publish** tab.
4. Choose **Publish function**.
5. Choose **Add association**. On the dialog box that appears, choose a distribution, an event type, and/or a cache behavior.

   For the event type, choose when you want this function to run:

   - **Viewer Request** – Run the function every time CloudFront receives a request.
   - **Viewer Response** – Run the function every time CloudFront returns a response.
6. To save the configuration, choose **Add association**.

CloudFront associates the distribution with the function. Wait a few minutes for the associated distribution to finish deploying. You can choose **View distribution** on the function details page to check the progress.

CLI

To associate a function with an existing cache behavior

1. Open a command line window.
2. Enter the following command to save the distribution configuration for the distribution whose cache behavior you want to associate with a function. This command saves the

distribution configuration to a file named `dist-config.yaml`. To use this command, do the following:

- Replace *`DistributionID`* with the distribution's ID.
- Run the command on one line. In the example, line breaks are provided to make the example more readable.

```
aws cloudfront get-distribution-config \
    --id DistributionID \
    --output yaml > dist-config.yaml
```

When the command is successful, the AWS CLI returns no output.

3. Open the file named `dist-config.yaml` that you created. Edit the file to make the following changes.

   a. Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   b. In the cache behavior, find the object named `FunctionAssociations`. Update this object to add a function association. The YAML syntax for a function association looks like the following example.

      - The following example shows a viewer request event type (trigger). To use a viewer response event type, replace `viewer-request` with `viewer-response`.
      - Replace *`arn:aws:cloudfront::111122223333:function/ExampleFunction`* with the Amazon Resource Name (ARN) of the function that you're associating with this cache behavior. To get the function ARN, you can use the **aws cloudfront list-functions** command.

```
FunctionAssociations:
  Items:
    - EventType: viewer-request
      FunctionARN: arn:aws:cloudfront::111122223333:function/ExampleFunction
  Quantity: 1
```

   c. After making these changes, save the file.

4. Use the following command to update the distribution, adding the function association. To use this command, do the following:

- Replace *DistributionID* with the distribution's ID.
- Run the command on one line. In the example, line breaks are provided to make the example more readable.

```
aws cloudfront update-distribution \
    --id DistributionID \
    --cli-input-yaml file://dist-config.yaml
```

When the command is successful, you see output like the following that describes the distribution that was just updated with the function association. The following example output is truncated for readability.

```
Distribution:
  ARN: arn:aws:cloudfront::111122223333:distribution/EBEDLT3BGRBBW
  ... truncated ...
  DistributionConfig:
    ... truncated ...
    DefaultCacheBehavior:
      ... truncated ...
      FunctionAssociations:
        Items:
        - EventType: viewer-request
          FunctionARN: arn:aws:cloudfront::111122223333:function/ExampleFunction
        Quantity: 1
      ... truncated ...
  DomainName: d111111abcdef8.cloudfront.net
  Id: EDFDVBD6EXAMPLE
  LastModifiedTime: '2021-04-19T22:39:09.158000+00:00'
  Status: InProgress
ETag: E2VJGGQEG1JT8S
```

The distribution's `Status` changes to `InProgress` while the distribution is redeployed. When the new distribution configuration reaches a CloudFront edge location, that edge location starts using the associated function. When the distribution is fully deployed, the `Status` changes back to `Deployed`. This indicates that the associated CloudFront function is live in all CloudFront edge locations worldwide. This typically takes a few minutes.

Amazon CloudFront KeyValueStore

CloudFront KeyValueStore is a secure, global, low-latency key value datastore that allows read access from within CloudFront Functions, enabling advanced customizable logic at the CloudFront edge locations.

With CloudFront KeyValueStore, you make updates to function code and updates to the data associated with a function independently of each other. This separation simplifies function code and makes it easy to update data without the need to deploy code changes.

> **Note**
>
> To use CloudFront KeyValueStore, your CloudFront function must use JavaScript runtime 2.0.

The following is the general procedure for using key-value pairs:

- Create key value stores, and populate it with a set of key-value pairs. You can add your key value stores to an Amazon S3 bucket or enter them manually.
- Associate the key value stores with your CloudFront function.
- Within your function code, use the name of the key to either retrieve the value associated with the key or to evaluate if a key exists. For more information about using key-value pairs in function code, and about helper methods, see the section called “Helper methods for key value stores”.

Use cases

You can use key-value pairs for the following examples:

- **URL rewrites or redirects** – The key-value pair can hold the rewritten URLs or the redirect URLs.
- **A/B testing and feature flags** – You can create a function to run experiments by assigning a percentage of traffic to a specific version of your website.
- **Access authorization** – You can implement access control to allow or deny requests based on criteria defined by you and the data stored in a key value store.

Supported formats for values

You can store the value in a key-value pair in any of the following formats:

- String
- Byte-encoded string
- JSON

Security

The CloudFront function and all its key value stores data are handled securely, as follows:

- CloudFront encrypts each key value stores at rest and during transit (when reading or writing to the key value stores) when you call the CloudFront KeyValueStore API operations.
- When the function is run, CloudFront decrypts each key-value pair in memory at the CloudFront edge locations.

To get started with CloudFront KeyValueStore, see the following topics.

Topics

- Work with key value store
- Work with key value data

- For more information about getting started with CloudFront KeyValueStore, see the Introducing Amazon CloudFront KeyValueStore AWS blog post.

Work with key value store

You must create a key value store to hold the key-value pairs that you want to use in CloudFront Functions.

After you create the key value stores and add key-value pairs, you can use the key values in your CloudFront function code.

To get started, see the following topics:

Topics

- Create a key value store
- Associate a key value store with a function
- Update a key value store
- Get a reference to a key value store
- Delete a key value store
- File format for key-value pairs

> **Note**
>
> The JavaScript runtime 2.0 includes some helper methods for working with key values in the function code. For more information, see the section called "Helper methods for key value stores".

Create a key value store

You can create a key value store and its key-value pairs at the same time. You can also create an empty key value store now and then add the key-value pairs later.

> **Note**
>
> If you specify your data source from an Amazon S3 bucket, you must have the `s3:GetObject` and `s3:GetBucketLocation` permissions to that bucket. If you don't have these permissions, CloudFront can't successfully create your key value store.

Decide if you want to add key-value pairs at the same time when you create the key value store. You can import your key-value pairs by using the CloudFront console, CloudFront API, or AWS SDKs. However, you can only import your file of key-value pairs when you *initially* create the key value store.

To create a file of key-value pairs, see File format for key-value pairs.

Console

To create a key value store

1. Sign in to the AWS Management Console and open the **Functions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions.
2. Choose the **KeyValueStores** tab, and then choose **Create KeyValueStore**.
3. Enter a name and optional description for the key value store.
4. Complete **S3 URI**:
   - If you have a file of key-value pairs, enter the path to the Amazon S3 bucket where you stored the file.
   - Leave this field blank if you plan to enter the key-value pairs manually.
5. Choose **Create**. The key value store now exists.

   The details page for the new key value store appears. The information on the page includes the ID and the ARN of the key value store.

   - The ID is a random string of characters that is unique in your AWS account.
   - The ARN has this syntax:

     ```
     AWS account:key-value-store/the key value stores ID
     ```
6. Look at the **Key value pairs** section. If you imported a file, this section shows some key-value pairs. You can do the following:
   - If you imported a file, you can also add more values manually.
   - If you didn't import a file from an Amazon S3 bucket, and if you want to add key-value pairs now, you can complete the next step.
   - You can skip this step and add the key-value pairs later.
7. To add the pairs now:
   a. Choose **Add key-value pairs**.
   b. Choose **Add pair** and enter a name and value. Repeat this step to add more pairs.
   c. When you're finished, choose **Save changes** to save all the key-value pairs in the key value store. On the dialog box that appears, choose **Done**.

8. To associate the key value store with a function now, complete the **Associated functions** section. For more information, see ??? or ???.

   You can also associate the function later, either from this key value store details page, or from the function's details page.

AWS CLI

To create a key value store

- Run the following command to create a key value store and import the key-value pairs from an Amazon S3 bucket.

```
aws cloudfront create-key-value-store \
    --name=keyvaluestore1 \
    --comment="This is my key value store file" \
    --import-source=SourceType=S3,SourceARN=arn:aws:s3:::amzn-s3-demo-
bucket1/kvs-input.json
```

Response

```
{
    "ETag": "ETVABCEXAMPLE",
    "Location": "https://cloudfront.amazonaws.com/2020-05-31/key-value-store/
arn:aws:cloudfront::123456789012:key-value-store/8aa76c93-3198-462c-aaf6-
example",
    "KeyValueStore": {
        "Name": "keyvaluestore1",
        "Id": "8aa76c93-3198-462c-aaf6-example",
        "Comment": "This is my key value store file",
        "ARN": "arn:aws:cloudfront::123456789012:key-value-
store/8aa76c93-3198-462c-aaf6-example",
        "Status": "PROVISIONING",
        "LastModifiedTime": "2024-08-06T22:19:10.813000+00:00"
    }
}
```

API

To create a key value store

1. Use the CloudFront CreateKeyValueStore operation. The operation takes several parameters:
   - A `name` of the key value store.
   - A `comment` parameter that includes a comment.
   - An `import-source` parameter that lets you import key-value pairs from a file that is stored in an Amazon S3 bucket. You can import from a file only when you first create the key value store. For information about the file structure, see the section called "File format for key-value pairs".

The operation response includes the following information:

- The values passed in the request, including the name that you assigned.
- Data such as the creation time.
- An `ETag` (for example, `ETVABCEXAMPLE`), the ARN that includes the name of the key value store (for example, `arn:aws:cloudfront::123456789012:key-value-store/keyvaluestore1`).

  You will use some combination of the `ETag`, the ARN, and the name to work with the key value store programmatically.

Key value store statuses

When you create a key value store, the data store can have the following status values.

Value	Description
Provisioning	The key value store was created and CloudFront is processing the data source that you specified.
Ready	The key value store was created and CloudFront successfully processed the data source that you specified.

Value	Description
Import failed	CloudFront wasn't able to process the data source that you specified. This status can appear if your file format isn't valid or that it exceeds the size limit. For more information, see File format for key-value pairs.

Associate a key value store with a function

After you create your key value store, you can update your function to associate it with your key value store. You must make this association to use the key-value pairs from that store in that function. The following rules apply:

- A function can have only one key value store
- You can associate the same key value store with multiple functions

Console

To associate a key value store with a function

1. Sign in to the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions and choose the **Functions** page.
2. Choose the function name.
3. Go to the **Associate KeyValueStore** section and choose **Associate existing KeyValueStore**.
4. Select the key value store that contains the key-value pairs in the function, and then choose **Associate KeyValueStore**.

   CloudFront immediately associates the store with the function. You don't need to save the function.
5. To specify a different key value store, choose **Update associated KeyValueStore**, select another key value store name, and then choose **Associate KeyValueStore**.

For more information, see the section called “Update functions”.

AWS CLI

To associate a key value store with a function

- Run the following command to update the *MaxAge* function and associate a key value store resource.

```
aws cloudfront update-function \
    --name MaxAge \
    --function-config '{"Comment":"Max Age 2 years","Runtime":"cloudfront-
js-2.0","KeyValueStoreAssociations":{"Quantity":1,"Items":
[{"KeyValueStoreARN":"arn:aws:cloudfront::123456789012:key-value-
store/8aa76c93-3198-462c-aaf6-example"}]}}' \
    --function-code fileb://function-max-age-v1.js \
    --if-match ETVABCEXAMPLE
```

- To associate a key value store with a function, specify the `KeyValueStoreAssociations` parameter and the key value store ARN.
- To change the association, specify another key value store ARN.
- To remove the association, remove the `KeyValueStoreAssociations` parameter.

For more information, see the section called "Update functions".

API

To associate a key value store with a function

- Use the UpdateFunction API operation. For more information, see the section called "Update functions".

Notes

- If you modify a key value store without changing the key-value pairs, or if you only modify the key-value pairs in the key value store, you don't need to associate the key value store again. You also don't need to republish the function.

  However, we recommend that you test the function to verify that it works as expected. For more information, see Test functions.

- You can view all the functions that use specific key value stores. On the CloudFront console, choose the key value store details page.

Update a key value store

When you update a key value store, you can change the key-value pairs, or change the association between the key value store and the function.

Console

To update a key value store

1. Sign in to the AWS Management Console and open the **Functions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions.
2. Choose the **KeyValueStores** tab.
3. Select the key value store that you want to update.
   - To update the key-value pairs, choose **Edit** in the **Key value pairs** section. You can add or delete any key-value pairs. You can also change the value for an existing key-value pair. When you're finished, choose **Save changes**.
   - To update the association for this key value store, choose **Go to functions**. For more information, see the section called “Associate a key value store with a function”.

AWS CLI

To update a key value store

1. **Change the key-value pairs** – You can add more key-value pairs, delete one or more key-value pairs, and change the value of an existing key-value pair. For more information, see Work with key value data.
2. **Change the function association for the key value store** – To update the function association for the key value store, see Associate a key value store with a function.

> **Tip**
>
> You will need the ARN of the key value store. For more information, see the section called "Get a reference to a key value store".

API

To update a key value store

1. **Change the key-value pairs** – You can add more key-value pairs, delete one or more key-value pairs, and change the value of an existing key-value pair. For more information, see Work with key value data.
2. **Change the function association for the key value store** – To update the function association for the key value store, use the UpdateFunction API operation. For more information, see the section called "Update functions".

> **Tip**
>
> You will need the ARN of the key value store. For more information, see the section called "Get a reference to a key value store".

Get a reference to a key value store

To work with the key value stores programmatically, you need the `ETag` and the name of the key value store.

To get both values, you can use the AWS Command Line Interface (AWS CLI) or the CloudFront API.

AWS CLI

To get the key value store reference

1. To return a list of key value stores, run the following command. Find the name of the key value store that you want to change.

```
aws cloudfront list-key-value-stores
```

2. From the response, find the name of the key value store that you want.

Response

```
{
    "KeyValueStoreList": {
        "Items": [
            {
                "Name": "keyvaluestore3",
                "Id": "37435e19-c205-4271-9e5c-example3",
                "ARN": "arn:aws:cloudfront::123456789012:key-value-
store/37435e19-c205-4271-9e5c-example3",
                "Status": "READY",
                "LastModifiedTime": "2024-05-08T14:50:18.876000+00:00"
            },
            {
                "Name": "keyvaluestore2",
                "Id": "47970d59-6408-474d-b850-example2",
                "ARN": "arn:aws:cloudfront::123456789012:key-value-
store/47970d59-6408-474d-b850-example2",
                "Status": "READY",
                "LastModifiedTime": "2024-05-30T21:06:22.113000+00:00"
            },
            {
                "Name": "keyvaluestore1",
                "Id": "8aa76c93-3198-462c-aaf6-example",
                "ARN": "arn:aws:cloudfront::123456789012:key-value-
store/8aa76c93-3198-462c-aaf6-example",
                "Status": "READY",
                "LastModifiedTime": "2024-08-06T22:19:30.510000+00:00"
            }
        ]
    }
}
```

3. Run the following command to return the ETag for the specified key value store.

```
aws cloudfront describe-key-value-store \
    --name=keyvaluestore1
```

Response

```
{
    "ETag": "E3UN6WX5RRO2AG",
    "KeyValueStore": {
        "Name": "keyvaluestore1",
        "Id": "8aa76c93-3198-462c-aaf6-example",
        "Comment": "This is an example KVS",
        "ARN": "arn:aws:cloudfront::123456789012:key-value-
store/8aa76c93-3198-462c-aaf6-example",
        "Status": "READY",
        "LastModifiedTime": "2024-08-06T22:19:30.510000+00:00"
    }
}
```

API

To get the key value store reference

1. Use the CloudFront ListKeyValueStores API operation to return a list of key value stores. Find the name of the key value store that you want to change.
2. Use the CloudFront DescribeKeyValueStore API operation and specify the name of the key value store that you returned from the previous step.

The response includes a UUID, the ARN of the key value store, and the `ETag` of the key value store.

- An `ETag`, such as `E3UN6WX5RRO2AG`
- The UUID is 128 bits, such as `8aa76c93-3198-462c-aaf6-example`
- The ARN includes the AWS account number, the constant `key-value-store`, and the UUID, like the following example:

  `arn:aws:cloudfront::123456789012:key-value-store/8aa76c93-3198-462c-aaf6-example`

For more information about the `DescribeKeyValueStore` operation, see the section called "About the CloudFront KeyValueStore".

Delete a key value store

You can delete your key value store by using the Amazon CloudFront console or API.

Console

To delete a key value store

1. Sign in to the AWS Management Console and open the **Functions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions.
2. Choose the function name.
3. Under the **Associated KeyValueStore** section, verify if a key value store is associated with the function. If it is, remove the association by choosing **Disassociate KeyValueStore** and then choose **Remove association**.
4. In the navigation pane, choose the **Functions** page and then choose the **KeyValueStores** tab.
5. Select the key value store that you want to delete and then choose **Delete**.

AWS CLI

To delete a key value store

1. Get the ETag and the name of the key value stores. For more information, see the section called “Get a reference to a key value store”.
2. Verify if the key value stores is associated with a function. If it is, remove the association. For more information on both these steps, see ???.
3. After you have the name and ETag of the key value store and it's no longer associated with a function, you can delete it.

   Run the following command to delete the specified key value store.

```
aws cloudfront delete-key-value-store \
    --name=keyvaluestore1 \
    --if-match=E3UN6WX5RRO2AG
```

API

To delete a key value store

1. Get the ETag and the name of the key value stores. For more information, see the section called “Get a reference to a key value store”.
2. Verify if the key value stores is associated with a function. If it is, remove the association. For more information on both these steps, see ???.
3. To delete the key value store, use the CloudFront DeleteKeyValueStore API operation.

File format for key-value pairs

When you create a UTF-8 encoded file, use the following JSON format:

```
{
  "data":[
    {
      "key":"key1",
      "value":"value"
    },
    {
      "key":"key2",
      "value":"value"
    }
  ]
}
```

Your file can't include duplicate keys. If you specified an invalid file in your Amazon S3 bucket, you can update the file to remove any duplicates and then try creating your key value store again.

For more information, see Create a key value store.

> **Note**
>
> The file for your data source and its key-value pairs have the following limits:
>
> - File size – 5 MB
> - Key size – 512 characters
> - Value size – 1024 characters

Work with key value data

This topic describes how to add key-value pairs to an existing key value store. To include key-value pairs when you initially create the key value stores, see the section called "Create a key value store".

Topics

- Work with key-value pairs (console)
- About the CloudFront KeyValueStore
- Work with key-value pairs (AWS CLI)
- Work with key-value pairs (API)

Work with key-value pairs (console)

You can use the CloudFront console to work with your key-value pairs.

To work with key-value pairs

1. Sign in to the AWS Management Console and open the **Functions** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home#/functions.
2. Choose the **KeyValueStores** tab.
3. Select the key value store that you want to change.
4. In the **Key value pairs** section, choose **Edit**.
5. You can add a key-value pair, delete a key-value pair, or change the value for an existing key-value pair.
6. When you're finished, choose **Save changes**.

About the CloudFront KeyValueStore

> **Tip**
>
> The CloudFront KeyValueStore API is a global service that uses Signature Version 4A (SigV4A) for authentication. Using temporary credentials with SigV4A requires version 2 session tokens. For more information, see Using temporary credentials with the CloudFront KeyValueStore API.

If you're using the AWS Command Line Interface (AWS CLI) or your own code to call the CloudFront KeyValueStore API, see the following sections.

When you work with a key value store and its key-value pairs, the service that you call depends on your use case:

- To work with key-value pairs in an *existing* key value store, use the CloudFront KeyValueStore service.
- To include key-value pairs in the key value store when you *initially* create the key value store, use the CloudFront service.

Both the CloudFront API and the CloudFront KeyValueStore API have a `DescribeKeyValueStore` operation. You call them for different reasons. To understand the differences, see the following table.

	CloudFront DescribeKeyValueStore API	**CloudFront KeyValueStore DescribeKeyValueStore API**
Data about the key value store	Returns data, such as the status and the date that the key value store itself was last modified.	Returns data about the *contents* of the storage resource – the key-value pairs in the store, and the size of the contents.
Data that identifies the key value store	Returns an `ETag`, the UUID, and the ARN of the key value store.	Returns an `ETag` and the ARN of the key value store.

Notes

- Each DescribeKeyValueStore operation returns a *different* `ETag`. The `ETags` aren't interchangeable.
- When you call an API operation to complete an action, you must specify the `ETag` from the appropriate API. For example, in the CloudFront KeyValueStore DeleteKey

operation, you specify the ETag that you returned from the CloudFront KeyValueStore DescribeKeyValueStore operation.

- When you invoke your CloudFront Functions by using CloudFront KeyValueStore, the values in the key value store aren't updated or changed during the invocation of the function. Updates are processed in between invocations of a function.

Work with key-value pairs (AWS CLI)

You can run the following AWS Command Line Interface commands for CloudFront KeyValueStore.

Contents

- List key-value pairs
- Get key-value pairs
- Describe a key value store
- Create a key-value pair
- Delete a key-value pair
- Update key-value pairs

List key-value pairs

To list key-value pairs in your key value store, run the following command.

```
aws cloudfront-keyvaluestore list-keys \
    --kvs-arn=arn:aws:cloudfront::123456789012:key-value-store/37435e19-c205-4271-9e5c-
example
```

Response

```
{
    "Items": [
        {
            "Key": "key1",
            "Value": "value1"
        }
    ]
}
```

Get key-value pairs

To get a key-value pair in your key value store, run the following command.

```
aws cloudfront-keyvaluestore get-key \
    --key=key1 \
    --kvs-arn=arn:aws:cloudfront::123456789012:key-value-store/37435e19-c205-4271-9e5c-
example
```

Response

```
{
    "Key": "key1",
    "Value": "value1",
    "ItemCount": 1,
    "TotalSizeInBytes": 11
}
```

Describe a key value store

To describe a key value store, run the following command.

```
aws cloudfront-keyvaluestore describe-key-value-store \
    --kvs-arn=arn:aws:cloudfront::123456789012:key-value-store/37435e19-c205-4271-9e5c-
example
```

Response

```
{
    "ETag": "KV1F83G8C2ARO7P",
    "ItemCount": 1,
    "TotalSizeInBytes": 11,
    "KvsARN": "arn:aws:cloudfront::123456789012:key-value-store/37435e19-
c205-4271-9e5c-example",
    "Created": "2024-05-08T07:48:45.381000-07:00",
    "LastModified": "2024-08-05T13:50:58.843000-07:00",
    "Status": "READY"
}
```

Create a key-value pair

To create a key-value pair in your key value store, run the following command.

```
aws cloudfront-keyvaluestore put-key \
    --if-match=KV1PA6795UKMFR9 \
    --key=key2 \
    --value=value2 \
    --kvs-arn=arn:aws:cloudfront::123456789012:key-value-store/37435e19-c205-4271-9e5c-
example
```

Response

```
{
    "ETag": "KV13V1IB3VIYZZH",
    "ItemCount": 3,
    "TotalSizeInBytes": 31
}
```

Delete a key-value pair

To delete a key-value pair, run the following command.

```
aws cloudfront-keyvaluestore delete-key \
    --if-match=KV13V1IB3VIYZZH \
    --key=key1 \
    --kvs-arn=arn:aws:cloudfront::123456789012:key-value-store/37435e19-c205-4271-9e5c-
example
```

Output

```
{
    "ETag": "KV1VC38T7YXB528",
    "ItemCount": 2,
    "TotalSizeInBytes": 22
}
```

Update key-value pairs

You can use the `update-keys` command to update more than one key-value pair. For example, to delete an existing key-value pair and create another one, run the following command.

```
aws cloudfront-keyvaluestore update-keys \
    --if-match=KV2EUQ1WTGCTBG2 \
    --kvs-arn=arn:aws:cloudfront::123456789012:key-value-store/37435e19-c205-4271-9e5c-
example \
```

```
    --deletes '[{"Key":"key2"}]' \
    --puts '[{"Key":"key3","Value":"value3"}]'
```

Response

```
{
    "ETag": "KV3AEGXETSR30VB",
    "ItemCount": 3,
    "TotalSizeInBytes": 28
}
```

Work with key-value pairs (API)

Follow this section to work with your key-value pairs programmatically.

Contents

- Get a reference to a key value store
- Change key-value pairs in a key value store
- Example code for CloudFront KeyValueStore

Get a reference to a key value store

When you use the CloudFront KeyValueStore API to call a write operation, you need to specify the ARN and the `ETag` of the key value store. To get this data, do the following:

To get a reference to a key value store

1. Use the CloudFront ListKeyValueStores API operation to get a list of key value stores. Find the key value store that you want to change.
2. Use the CloudFrontKeyValueStore DescribeKeyValueStore API operation and specify the key value store from the previous step.

   The response includes the ARN and the `ETag` of the key value store.

   - The ARN includes the AWS account number, the constant `key-value-store`, and the UUID, such as the following example:

     `arn:aws:cloudfront::123456789012:key-value-store/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111`

- An `ETag` that looks like the following example:

  `ETVABCEXAMPLE2`

Change key-value pairs in a key value store

You can specify the key value store that contains the key-value pair that you want to update.

See the following CloudFront KeyValueStore API operations:

- CloudFrontKeyValueStore DeleteKey – Deletes a key-value pair
- CloudFrontKeyValueStore GetKey – Returns a key-value pair
- CloudFrontKeyValueStore ListKeys – Returns a list of key-value pairs
- CloudFrontKeyValueStore PutKey – You can perform the following tasks:
  - Create a key-value pair in one key value store by specifying a new key name and value.
  - Set a different value in an existing key-value pair by specifying an existing key name, and a new key value.
- CloudFrontKeyValueStore UpdateKeys – You can perform one or more of the following actions in one all-or-nothing operation:
  - Delete one or more key-value pairs
  - Create one or more new key-value pairs
  - Set a different value in one or more existing key-value pairs

Example code for CloudFront KeyValueStore

Example

The following code shows you how to call the `DescribeKeyValueStore` API operation for a key value store.

```
const {
  CloudFrontKeyValueStoreClient,
  DescribeKeyValueStoreCommand,
} = require("@aws-sdk/client-cloudfront-keyvaluestore");

require("@aws-sdk/signature-v4-crt");
```

```
(async () => {
  try {
    const client = new CloudFrontKeyValueStoreClient({
      region: "us-east-1"
    });
    const input = {
      KvsARN: "arn:aws:cloudfront::123456789012:key-value-store/a1b2c3d4-5678-90ab-
cdef-EXAMPLE11111",
    };
    const command = new DescribeKeyValueStoreCommand(input);

    const response = await client.send(command);
  } catch (e) {
    console.log(e);
  }
})();
```

Customize with CloudFront Connection Functions

CloudFront Connection Functions allow you to write lightweight JavaScript functions for mTLS certificate validation and custom authentication logic. Your Connection Functions run during mTLS connection establishment to validate client certificates, implement device-specific authentication rules, and handle certificate revocation scenarios. The Connection Functions runtime environment offers submillisecond startup times, scales immediately to handle millions of connections per second, and is highly secure. Connection Functions are a native feature of CloudFront, which means you can build, test, and deploy your code entirely within CloudFront.

When you associate a Connection Function with an mTLS-enabled CloudFront distribution, CloudFront intercepts TLS connection requests at CloudFront edge locations and passes certificate information to your function. You can invoke Connection Functions when the following event occurs:

- During TLS connection establishment (connection request) - for mutual TLS (mTLS) connections

For more information about Connection Functions, see the following topics.

Topics

- Overview and workflow
- Configuration and limits

- Create CloudFront Connection Functions for mutual TLS (viewer) validation
- Write CloudFront Connection Function code for mutual TLS (viewer) validation
- Test CloudFront Connection Functions before deployment
- Associate Connection Functions with distributions
- Implement certificate revocation for mutual TLS (viewer) with CloudFront Functions and KeyValueStore

Overview and workflow

CloudFront Connection Functions are a specialized type of CloudFront Functions that run during the TLS handshake when a client attempts to establish an mTLS connection. Your Connection Function can access client certificate information, mTLS configuration parameters, certificate revocation check results, and the client IP address.

Connection functions are invoked after CloudFront performs standard certificate validation (trust chain, expiry, signature verification) but can run even if certificate revocation checks fail. With this, you can implement custom logic for handling revoked certificates or adding additional validation criteria.

After you create and publish a Connection Function, make sure to add an association for the connection request event type with an mTLS-enabled distribution. This makes the function run each time a client attempts to establish an mTLS connection with CloudFront.

CloudFront Connection Functions follow a two-stage lifecycle that you can use to develop and test functions before deploying them to production. This workflow makes sure that your Connection Functions work correctly before they affect live traffic.

Topics

- Function stages
- Development workflow
- Differences from other function types

Function stages

Connection functions exist in one of two stages:

- **DEVELOPMENT** – Functions in this stage can be modified, tested, and updated. Use this stage for writing and debugging your function code.
- **LIVE** – Functions in this stage are read-only and handle production traffic. You cannot modify functions in the LIVE stage directly.

When you create a new Connection Function, it starts in the **DEVELOPMENT** stage. After testing and validation, you publish the function to move it to the **LIVE** stage.

Development workflow

Follow this workflow to develop and deploy Connection Functions:

1. **Create** – Create a new Connection Function in the DEVELOPMENT stage with your initial code and configuration.
2. **Test** – Use the test functionality to validate your function with sample connection events before deployment.
3. **Update** – Modify the function code and configuration as needed based on test results.
4. **Publish** – When ready for production, publish the function to move it from DEVELOPMENT to LIVE stage.
5. **Associate** – Associate the published function with your mTLS-enabled distribution to handle live connections.

To make changes to a LIVE function, you must update the DEVELOPMENT version and publish it again. This creates a new version in the LIVE stage.

Differences from other function types

Connection functions differ from viewer request and viewer response functions in several important ways:

- Connection functions run after the mTLS handshake, before any HTTP processing occurs
- Connection functions have access to TLS certificate information instead of HTTP request/response data
- Connection functions can only allow or deny the connection, not modify HTTP data
- Connection functions are only invoked for new TLS connections, not for connection reuse

- TLS session resumption is not supported with mTLS to make sure certificate validation occurs on every connection
- Connection functions run in addition to standard viewer request and viewer response functions
- You associate Connection Functions at the distribution level, instead of at the cache behavior level.
- Connection functions only support JavaScript runtime 2.0.

Configuration and limits

CloudFront Connection Functions have specific configuration requirements and service limits due to their specialized role in TLS connection validation and the performance requirements of edge computing.

Topics

- Function code requirements
- Service limits
- Function filtering options

Function code requirements

Connection functions require JavaScript code that processes TLS connection events. The function code must:

- Be written in JavaScript
- Process connection events and make allow/deny decisions
- Complete execution within the time limits
- Handle certificate and connection validation logic

Service limits

Connection functions are subject to the following limits:

- **Function size** – Function code and configuration are limited in size
- **Execution time** – Functions have strict execution time limits for TLS connection processing
- **Association limits** – Each distribution can have only one Connection Function associated

- **Stage restrictions** – Only LIVE stage functions can be associated with distributions

Function filtering options

When listing Connection Functions, you can use the following filters:

- **Stage filter** – Filter by DEVELOPMENT or LIVE stage
- **Association filter** – Filter by distribution ID or key-value store ID associations

These filters help you organize and manage Connection Functions across different environments and use cases.

Create CloudFront Connection Functions for mutual TLS (viewer) validation

You create a CloudFront Connection Function in two stages:

1. Create the function code as JavaScript. You can use the default example from the CloudFront console or write your own. For more information, see the following topics:
   - Write CloudFront Connection Function code for mTLS validation
   - CloudFront Connection Function event structure and response format
   - Connection function code examples
2. Use CloudFront to create the Connection Function and include your code. The code exists inside the function (not as a reference).

Topics

- CloudFront console
- AWS CLI

CloudFront console

To create a Connection Function

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. Choose **Create function**.
3. Enter a function name that is unique within the AWS account, choose **Connection function** as the function type, and then choose **Continue**.
4. The details page for the new Connection Function appears.

> **Note**
>
> Connection functions only support JavaScript runtime 2.0. To use CloudFront Connection Function KeyValueStore integration in the function, you must use this runtime version.

5. In the **Function code** section, choose the **Build** tab and enter your connection function code. The sample code that is included in the Build tab illustrates the basic syntax for Connection Function code.
6. Choose **Save changes**.
7. If the Connection Function code uses KeyValueStore for certificate revocation checking or device validation, you must associate a KeyValueStore.

   You can associate the KeyValueStore when you first create the function. Or, you can associate it later, by associating Connection Functions.

   To associate a KeyValueStore now, follow these steps:

   - Go to the **Associate KeyValueStore** section and choose **Associate existing KeyValueStore**.
   - Select the KeyValueStore that contains the certificate data for your Connection Function, and then choose **Associate KeyValueStore**.

   CloudFront immediately associates the store with the function. You don't need to save the function.

AWS CLI

If you use the AWS CLI, you typically first create the Connection Function code in a file, and then create the function with the AWS CLI.

To create a Connection Function

1. Create the Connection Function code in a file, and store it in a directory that your computer can connect to.
2. Run the command as shown in the example. This example uses the `fileb://` notation to pass in the file. It also includes line breaks to make the command more readable.

```
aws cloudfront create-connection-function \
    --name CertificateValidator \
    --connection-function-config '{
        "Comment":"Device certificate validation",
        "Runtime":"cloudfront-js-2.0",
        "KeyValueStoreAssociations":{
            "Quantity":1,
            "Items":[{
                "KeyValueStoreARN":"arn:aws:cloudfront::111122223333:key-value-
store/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
            }]
        }
    }' \
    --connection-function-code fileb://certificate-validator.js
```

> **Note**
>
> - **Runtime** – Connection functions only support JavaScript runtime 2.0 (cloudfront-js-2.0).
> - **KeyValueStoreAssociations** – If your Connection Function uses KeyValueStore for certificate validation, you can associate the KeyValueStore when you first create the function. Or, you can associate it later, by using update-connection-function. The Quantity is always 1 because each Connection Function can have only one KeyValueStore associated with it.

3. When the command is successful, you see output like the following.

```
ETag: ETVABCEXAMPLE
ConnectionFunctionSummary:
  ConnectionFunctionConfig:
    Comment: Device certificate validation
    Runtime: cloudfront-js-2.0
```

```
    KeyValueStoreAssociations:
      Quantity: 1
      Items:
        - KeyValueStoreARN: arn:aws:cloudfront::111122223333:key-value-store/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
  ConnectionFunctionMetadata:
    CreatedTime: '2024-09-04T16:32:54.292000+00:00'
    ConnectionFunctionARN: arn:aws:cloudfront::111122223333:connection-function/
CertificateValidator
    LastModifiedTime: '2024-09-04T16:32:54.292000+00:00'
    Stage: DEVELOPMENT
  Name: CertificateValidator
  Status: UNPUBLISHED
Location: https://cloudfront.amazonaws.com/2020-05-31/connection-function/
arn:aws:cloudfront:::connection-function/CertificateValidator
```

Most of the information is repeated from the request. Other information is added by CloudFront.

> **Note**
>
> - **ETag** – This value changes each time you modify the Connection Function. You need this value to update or publish the function.
> - **Stage** – New connection functions start in the DEVELOPMENT stage. You must publish the function to move it to the LIVE stage before associating it with a distribution.
> - **Status** – The function status is UNPUBLISHED until you publish it to the LIVE stage.

Write CloudFront Connection Function code for mutual TLS (viewer) validation

CloudFront Connection Functions enable you to write lightweight JavaScript functions for mTLS certificate validation and custom authentication logic. Your Connection Function code can validate client certificates, implement device-specific authentication rules, handle certificate revocation scenarios, and make allow/deny decisions for TLS connections at CloudFront edge locations worldwide.

Connection functions provide a powerful way to extend CloudFront built-in certificate validation with your own business logic. Unlike viewer request and viewer response functions that process HTTP data, Connection Functions operate at the TLS layer and have access to certificate information, client IP addresses, and TLS connection details. This makes them ideal for implementing zero-trust security models, device authentication systems, and custom certificate validation policies that go beyond standard PKI validation.

Your Connection Function code runs in a secure, isolated environment with submillisecond startup times and can scale to handle millions of connections per second. The runtime is optimized for certificate validation workloads and provides built-in integration with CloudFront KeyValueStore for real-time data lookup operations, enabling sophisticated authentication scenarios like certificate revocation list checking and device allowlist validation.

To help you write effective Connection Function code, see the following topics. For complete code examples and step-by-step tutorials, see the tutorial sections in this guide and explore the Connection Function examples available in the CloudFront console.

Topics

- CloudFront Connection Function use cases and purposes
- CloudFront Connection Function event structure and response format
- CloudFront Connection Functions JavaScript runtime features
- CloudFront Connection Function helper methods and APIs
- CloudFront Connection Function KeyValueStore integration
- Use async and await
- Connection function code examples

CloudFront Connection Function use cases and purposes

Before writing your CloudFront Connection Function, carefully determine what type of certificate validation or authentication logic you need to implement. Connection functions are designed for specific use cases that require custom validation beyond standard PKI certificate checking. Understanding your use case helps you design efficient code that meets your security requirements while maintaining optimal performance.

Common Connection Function use cases include:

- **Certificate revocation handling** – Implement custom policies for handling revoked certificates, including grace periods for certificate rotation, trusted network exceptions for internal devices, or emergency access scenarios where revoked certificates might need temporary access.
- **Optional mTLS support** – Handle both mTLS and non-mTLS connections with different authentication policies, allowing you to provide enhanced security for clients that support certificates while maintaining compatibility with legacy clients.
- **IP-based authentication** – Combine certificate validation with client IP address checks for enhanced security, such as restricting access from specific geographic regions, corporate networks, or known malicious IP ranges.
- **Multi-tenant certificate validation** – Implement tenant-specific validation rules where different certificate authorities or validation criteria apply based on the client certificate issuer or subject attributes.
- **Time-based access control** – Enforce time-based restrictions where certificates are only valid during specific hours, maintenance windows, or business periods, even if the certificate itself hasn't expired.

Connection functions run after CloudFront performs standard certificate validation (trust chain verification, expiry checks, and signature validation) but before the TLS connection is established. This timing gives you the flexibility to add custom validation criteria while benefiting from CloudFront built-in certificate validation. Your function receives the results of standard validation and can make informed decisions about whether to allow or deny the connection based on both standard and custom criteria.

When designing your Connection Function, consider the performance implications of your validation logic. Functions have a 5-millisecond execution limit, so complex operations should be optimized for speed. Use KeyValueStore for fast data lookups rather than complex calculations, and structure your validation logic to fail fast for invalid certificates.

CloudFront Connection Function event structure and response format

CloudFront Connection Functions receive a different event structure than viewer request and viewer response functions. Instead of HTTP request/response data, connection functions receive certificate and connection information that you can use to make authentication decisions.

Topics

- Event structure for Connection Functions

- Connection Functions response format

Event structure for Connection Functions

Connection Functions receive an event object that contains certificate and connection information. The event structure of the function is shown below:

```
{
  "clientCertificate": {
    "certificates": {
      "leaf": {
        "serialNumber": "string",
        "issuer": "string",
        "subject": "string",
        "validity": {
          "notBefore": "string",
          "notAfter": "string"
        },
        "sha256Fingerprint": "string",
        "ocspEndpoint": "string",
        "san": "string",
        "spki": "string",
        "sanHex": "string"
      },
      "intermediates": [
        {
          "serialNumber": "string",
          "issuer": "string",
          "subject": "string",
          "validity": {
            "notBefore": "string",
            "notAfter": "string"
          },
          "sha256Fingerprint": "string",
          "ocspEndpoint": "string",
          "san": "string",
          "spki": "string",
          "sanHex": "string"
        }
      ]
    }
  },
  "clientIp": "string",
```

```
  "endpoint": "string",
  "distributionId": "string",
  "connectionId": "string"
}
```

Below is an example of the event object structure:

```
{
  "clientCertificate": {
    "certificates": {
      "leaf": {
        "serialNumber": "00:9e:2a:af:16:56:e5:47:25:7d:2e:38:c3:f9:9d:57:fa",
        "issuer": "C=US, O=Ram, OU=Edge, ST=WA, CN=mTLS-CA, L=Snoqualmie",
        "subject": "C=US, O=Ram, OU=Edge, ST=WA, CN=mTLS-CA, L=Snoqualmie",
        "validity": {
          "notBefore": "2025-09-10T23:43:10Z",
          "notAfter": "2055-09-11T00:43:02Z"
        },
        "sha256Fingerprint": "_w6bJ7aOAlGOj7NUhJxTfsfee-ONg_xop3_PTgTJpqs=",
        "ocspEndpoint": "http://ocsp.example.com",
        "san": "DNS:api.example.com,DNS:www.example.com,URI:urn:example:client:123,IP
 Address:192.168.1.100,email:alejandro_rosalez@example.com",
        "spki":
 "30820122300d06092a864886f70d0101010500038201 0f003082010a0282010100b8efe87db798500320936...",
        "sanHex":
 "3058820f6170692e6578616d706c652e636f6d820f7777772e6578616d706c652e636f6d8619..."
      },
      "intermediates": [
        {
          "serialNumber": "00:d4:04:a5:fd:cb:20:37:54",
          "issuer": "C=US, O=Example Corp, CN=Example Root CA",
          "subject": "C=US, O=Example Corp, CN=Example Intermediate CA",
          "validity": {
            "notBefore": "2024-01-01T00:00:00Z",
            "notAfter": "2034-01-01T23:59:59Z"
          },
          "sha256Fingerprint": "3a8c6b...",
          "ocspEndpoint": "http://ocsp.example.com/intermediate",
          "san": "DNS:ca.example.com,URI:urn:ca:intermediate:001",
          "spki":
 "30820122300d06092a864886f70d0101010500038201 0f003082010a0282010100b389dd513b0f1f14cac1...",
          "sanHex": "302b820e63612e6578616d706c652e636f6d..."
        }
```

```
      ]
    }
  },
  "clientIp": "192.0.2.1",
  "endpoint": "d3lch071jze0cb.cloudfront.net",
  "distributionId": "E1NXS4MQZH501R",
  "connectionId": "NpvTe1925xfj24a67sPQr7ae42BIq03FGhJJKfrQYWZcWZFp96SIIg=="
}
```

Connection Functions response format

Your Connection Function must return a response object that indicates whether to allow or deny the connection. Use the helper methods to make connection decisions:

```
function connectionHandler(connection) {
    // Helper methods to allow or deny connections
    if (/* some logic to determine if function should allow connection */) {
        connection.allow();
    } else {
        connection.deny();
    }
}
```

Unlike viewer request and viewer response functions, Connection Functions cannot modify HTTP requests or responses. They can only allow or deny the TLS connection.

CloudFront Connection Functions JavaScript runtime features

CloudFront Connection Functions use the CloudFront Functions JavaScript runtime 2.0, which provides a secure and high-performance environment specifically optimized for certificate validation workloads. The runtime is designed to start in sub-milliseconds and handle millions of concurrent executions across CloudFront global edge network.

The runtime environment includes comprehensive JavaScript language support:

- **ECMAScript 2020 (ES11) support** – Modern JavaScript features including optional chaining (?.) and nullish coalescing (??)
- **Built-in objects** – Standard JavaScript objects like Object, Array, JSON, Math, and Date
- **Console logging** – Use console.log() for debugging and monitoring certificate validation decisions. Logs are available in real-time during testing and can help troubleshoot validation logic in development

- **KeyValueStore integration** – Native access to CloudFront KeyValueStore for ultra-fast data lookup operations, enabling real-time certificate revocation checking, device allowlist validation, and tenant-specific configuration retrieval

Connection functions are optimized for high-performance for certificate validation scenarios. The runtime automatically handles memory management, garbage collection, and resource cleanup to make sure consistent performance across millions of concurrent connections. All operations are designed to be deterministic and fast, with KeyValueStore lookups typically completing in microseconds.

The runtime environment is completely isolated between function executions, making sure that no data leaks between different client connections. Each function execution starts with a clean state and has no access to previous execution results or client data from other connections.

CloudFront Connection Function helper methods and APIs

CloudFront Connection Functions provide specialized helper methods designed to simplify certificate validation decisions and enhance observability. These methods are optimized for the connection validation workflow and integrate seamlessly with CloudFront connection logging and monitoring systems.

- **connection.allow()** – Allow the TLS connection to proceed. This method signals CloudFront to complete the TLS handshake and allow the client to establish the connection. Use this when certificate validation passes and any custom authentication logic is satisfied
- **connection.deny()** – Deny the TLS connection and terminate the handshake. This method immediately closes the connection and prevents any HTTP traffic from flowing. The client will receive a TLS connection error. Use this for invalid certificates, failed authentication, or policy violations
- **connection.logCustomData()** – Add custom data to connection logs (up to 800 bytes of UTF-8 text). This method allows you to include validation results, certificate details, or decision rationale in CloudFront connection logs for security monitoring, compliance auditing, and troubleshooting

These methods provide a clean, declarative interface for making connection decisions and logging relevant information for monitoring and debugging. The allow/deny pattern makes sure that your function's intent is clear and that CloudFront can optimize connection handling based on your

decision. Custom logging data is immediately available in CloudFront connection logs and can be used with log analysis tools for security monitoring and operational insights.

Always call either connection.allow() or connection.deny() before your function completes. If neither method is called, CloudFront will deny the connection by default as a security precaution.

CloudFront Connection Function KeyValueStore integration

CloudFront Connection Functions can use CloudFront KeyValueStore to perform ultra-fast data lookups for certificate validation scenarios. KeyValueStore is particularly powerful for Connection Functions because it provides global, eventually-consistent data access with microsecond lookup times across all CloudFront edge locations. This makes it ideal for maintaining certificate revocation lists, device allowlists, tenant configurations, and other validation data that needs to be accessible during TLS handshakes.

KeyValueStore integration is designed specifically for high-performance connection validation workflows:

- **kvsHandle.exists(key)** – Check if a key exists in the KeyValueStore without retrieving the value. This is the most efficient method for binary validation scenarios like certificate revocation checking, where you only need to know if a certificate serial number is in a revocation list
- **kvsHandle.get(key)** – Retrieve a value from the KeyValueStore for more complex validation scenarios. Use this when you need to access configuration data, validation rules, or metadata associated with a certificate or device identifier

KeyValueStore operations are asynchronous and must be used with async/await syntax. The KeyValueStore has a 10MB total size limit and supports up to 10 million key-value pairs. KeyValueStore data is eventually consistent across all edge locations, with updates typically propagating within seconds.

For optimal performance, structure your KeyValueStore keys to minimize lookup operations. Use certificate serial numbers as keys for simple revocation checking, or create composite keys combining issuer hash and serial number for multi-CA environments. Consider the trade-offs between key complexity and KeyValueStore capacity when designing your data structure.

Use async and await

Connection functions support asynchronous operations using async/await syntax, which is essential when working with KeyValueStore operations or other asynchronous tasks. The async/

await pattern makes sure that your function waits for KeyValueStore lookups to complete before making connection decisions, while maintaining the high-performance characteristics required for TLS handshake processing.

Proper async/await usage is critical for Connection Functions because KeyValueStore operations, while very fast, are still network operations that require coordination across CloudFront distributed infrastructure. The runtime automatically handles promise resolution and makes sure that your function completes within the 5-millisecond execution limit.

Example: Async Connection Function with KeyValueStore

```
import cf from 'cloudfront';

async function connectionHandler(connection) {
    const kvsHandle = cf.kvs();

    // Async operation to check KeyValueStore for certificate revocation
    const isRevoked = await
 kvsHandle.exists(connection.clientCertificate.certificates.leaf.serialNumber);

    if (isRevoked) {
        // Log the revocation decision with certificate details
        connection.logCustomData(`REVOKED_CERT:
${connection.clientCertificate.certificates.leaf.serialNumber}:
${connection.clientCertificate.certificates.leaf.issuer}`);
        console.log(`Denying connection for revoked certificate:
 ${connection.clientCertificate.certificates.leaf.serialNumber}`);
        return connection.deny();
    }

    // Log successful validation for monitoring
    connection.logCustomData(`VALID_CERT:
${connection.clientCertificate.certificates.leaf.serialNumber}`);
    console.log(`Allowing connection for valid certificate:
 ${connection.clientCertificate.certificates.leaf.serialNumber}`);
    return connection.allow();
}
```

Always use async/await when calling KeyValueStore methods or other asynchronous operations. The Connection Function runtime handles promise resolution automatically and makes sure proper execution flow within the strict timing constraints of TLS handshake processing. Avoid using .then()

or callback patterns, as async/await provides cleaner error handling and better performance in the Connection Function environment.

When designing async Connection Functions, structure your code to minimize the number of KeyValueStore operations and perform them as early as possible in your validation logic. This makes sure maximum performance and reduces the risk of timeout issues during high-traffic periods. Consider batching related validation checks and using the most efficient KeyValueStore method (exists() vs get()) for your use case.

Connection function code examples

The following examples demonstrate common Connection Function patterns for different validation scenarios. Use these examples as starting points for your own Connection Function implementations.

Example: Device certificate validation

This example validates device serial numbers and certificate subject fields for IoT devices, gaming consoles, and other client authentication scenarios:

```
async function connectionHandler(connection) {
    // Custom validation: check device serial number format
    const serialNumber = connection.clientCertificate.certificates.leaf.serialNumber;
    if (!serialNumber.startsWith("DEV")) {
        connection.logCustomData(`INVALID_SERIAL:${serialNumber}`);
        return connection.deny();
    }

    // Validate certificate subject contains required organizational unit
    const subject = connection.clientCertificate.certificates.leaf.subject;
    if (!subject.includes("OU=AuthorizedDevices")) {
        connection.logCustomData(`INVALID_OU:${subject}`);
        return connection.deny();
    }

    // Allow connection for valid devices
    connection.logCustomData(`VALID_DEVICE:${serialNumber}`);
    return connection.allow();
}
```

This function performs multiple validation checks beyond standard certificate validation, including device serial number format and organizational unit verification.

Example: Optional mTLS with mixed authentication

This example handles both mTLS and non-mTLS connections with different authentication policies:

```
async function connectionHandler(connection) {
    if (connection.clientCertificate) {
        // mTLS connection - enhanced validation for certificate holders
        const subject = connection.clientCertificate.certificates.leaf.subject;
        connection.logCustomData(`MTLS_SUCCESS:${subject}:${connection.clientIp}`);
        console.log(`mTLS connection from: ${subject}`);
        return connection.allow();
    } else {
        // Non-mTLS connection - apply IP-based restrictions
        const clientIp = connection.clientIp;

        // Only allow non-mTLS from specific IP ranges
        if (clientIp.startsWith("203.0.113.") || clientIp.startsWith("198.51.100.")) {
            connection.logCustomData(`NON_MTLS_ALLOWED:${clientIp}`);
            console.log(`Non-mTLS connection allowed from: ${clientIp}`);
            return connection.allow();
        }

        connection.logCustomData(`NON_MTLS_DENIED:${clientIp}`);
        return connection.deny();
    }
}
```

This function provides enhanced security for clients with certificates while maintaining compatibility with legacy clients from trusted IP ranges.

Test CloudFront Connection Functions before deployment

You can test CloudFront Connection Functions in the DEVELOPMENT stage using the TestConnectionFunction API operation. With testing, you can validate your function logic with sample connection events before publishing to the LIVE stage.

Topics

- Testing process
- Test results
- Connection test object

Testing process

To test a Connection Function:

1. Create a Connection Function in the DEVELOPMENT stage
2. Prepare a test connection object that represents the TLS connection event
3. Use the TestConnectionFunction API operation to execute your function with the test data
4. Review the test results, including function output, execution logs, and any error messages
5. Update your function code as needed and repeat the testing process

Test results

When you test a Connection Function, the results include:

- **Function summary** – Metadata about the function that was tested
- **Compute utilization** – Performance metrics showing resource usage
- **Execution logs** – Console output from your function, including any logging statements
- **Function output** – The result returned by your function
- **Error messages** – Any runtime errors or exceptions that occurred during execution

Connection test object

The connection test object is a binary blob (up to 40KB) that represents the TLS connection event your function will process. This object contains the certificate and connection information that your function uses to make authentication decisions.

> **Note**
>
> The specific structure and format of the connection test object is defined by the CloudFront Connection Functions runtime. Consult the CloudFront Functions documentation or contact AWS Support for details on creating appropriate test objects for your use case.

After creating your Connection Function, you can:

- **Test the function** – Use the test functionality in the console or CLI to validate your function with sample connection events. For more information, see Connection Function testing.

- **Update the function** – Modify the function code and configuration as needed. Connection functions in the DEVELOPMENT stage can be updated at any time.
- **Publish the function** – When ready for production, publish the function to move it from DEVELOPMENT to LIVE stage. For more information, see associating Connection Functions.
- **Associate with a distribution** – Associate the published function with an mTLS-enabled distribution to handle live connections. For more information, see associating connection functions.

Associate Connection Functions with distributions

After publishing a Connection Function to the LIVE stage, you must associate it with an mTLS-enabled distribution to handle live connections. Connection functions are associated at the distribution level, unlike viewer request and viewer response functions which are associated with cache behaviors.

Topics

- Association requirements
- Organizing functions with filters
- Deployment considerations

Association requirements

To associate a Connection Function with a distribution:

- The function must be in the LIVE stage
- The distribution must have mTLS enabled
- The distribution must have a valid trust store configured
- You can only associate one Connection Function per distribution

Organizing functions with filters

CloudFront provides filtering capabilities to help you organize and manage connection functions:

- **Distribution ID filter** – Find functions associated with specific distributions
- **Key-value store filter** – Find functions that use specific key-value stores for data lookup

- **Stage filter** – List functions in DEVELOPMENT or LIVE stage

Use these filters when managing multiple Connection Functions across different distributions or development environments.

Deployment considerations

Consider these factors when deploying Connection Functions:

- **Global deployment** – Connection functions deploy to all CloudFront edge locations worldwide, which might take several minutes
- **Version management** – Each published version creates a new LIVE function that replaces the previous version
- **Rollback strategy** – Plan for rollback by keeping previous working versions of your function code
- **Testing in production** – Consider using separate distributions for staging and production environments

Implement certificate revocation for mutual TLS (viewer) with CloudFront Functions and KeyValueStore

You can use CloudFront Connection Functions with KeyValueStore to implement certificate revocation checking. With this, you can maintain a list of revoked certificate serial numbers and check client certificates against this list during the TLS handshake.

To implement certificate revocation, you need these components:

- A distribution configured with viewer mTLS
- A KeyValueStore containing revoked certificate serial numbers
- A Connection Function that queries the KeyValueStore to check certificate status

When a client connects, CloudFront validates the certificate against the trust store, then runs your Connection Function. Your function checks the certificate serial number against the KeyValueStore and allows or denies the connection.

Topics

- Step 1: Create a KeyValueStore for revoked certificates

- Step 2: Create the revocation Connection Function
- Step 3: Test your revocation function
- Step 4: Associate the function to your distribution
- Advanced revocation scenarios

Step 1: Create a KeyValueStore for revoked certificates

Create a KeyValueStore to store revoked certificate serial numbers that your Connection Function can check during mTLS connections.

First, prepare your revoked certificate serial numbers in JSON format:

```
{
  "data": [
    {
      "key": "ABC123DEF456",
      "value": ""
    },
    {
      "key": "789XYZ012GHI",
      "value": ""
    }
  ]
}
```

Upload this JSON file to an S3 bucket, then create the KeyValueStore:

```
aws s3 cp revoked-serials.json s3://your-bucket-name/revoked-serials.json
aws cloudfront create-key-value-store \
  --name revoked-serials-kvs \
  --import-source '{
    "SourceType": "S3",
    "SourceARN": "arn:aws:s3:::your-bucket-name/revoked-serials.json"
  }'
```

Wait for the KeyValueStore to finish provisioning. Check the status with:

```
aws cloudfront get-key-value-store --name "revoked-serials-kvs"
```

Step 2: Create the revocation Connection Function

Create a Connection Function that checks certificate serial numbers against the KeyValueStore to determine if certificates are revoked.

Create a Connection Function that checks certificate serial numbers against the KeyValueStore:

```
aws cloudfront create-connection-function \
  --name "revocation-control" \
  --connection-function-config file://connection-function-config.json \
  --connection-function-code file://connection-function-code.txt
```

The configuration file specifies the KeyValueStore association:

```
{
  "Runtime": "cloudfront-js-2.0",
  "Comment": "A function that implements revocation control via KVS",
  "KeyValueStoreAssociations": {
    "Quantity": 1,
    "Items": [
      {
        "KeyValueStoreArn": "arn:aws:cloudfront::account-id:key-value-store/kvs-id"
      }
    ]
  }
}
```

The Connection Function code checks the KeyValueStore for revoked certificates:

```
import cf from 'cloudfront';

async function connectionHandler(connection) {
    const kvsHandle = cf.kvs();

    // Get parsed client serial number from client certificate
    const clientSerialNumber = connection.clientCertInfo.serialNumber;

    // Check KVS to see if serial number exists as a key
    const serialNumberExistsInKvs = await kvsHandle.exists(clientSerialNumber);

    // Deny connection if serial number exists in KVS
    if (serialNumberExistsInKvs) {
```

```
        console.log("Connection denied - certificate revoked");
        return connection.deny();
    }

    // Allow connections that don't exist in kvs
    console.log("Connection allowed");
    return connection.allow();
}
```

Step 3: Test your revocation function

Use the CloudFront console to test your Connection Function with sample certificates. Navigate to the Connection Function in the console and use the Test tab.

Test with sample certificates

1. Paste a sample certificate in PEM format into the test interface
2. Optionally specify a client IP address for testing IP-based logic
3. Choose **Test function** to see the execution results
4. Review the execution logs to verify your function logic

Test with both valid and revoked certificates to make sure your function handles both scenarios correctly. The execution logs show console.log output and any errors that occur during function execution.

Step 4: Associate the function to your distribution

After you publish your Connection Function, associate it with your mTLS-enabled distribution to activate certificate revocation checking.

You can associate the function from the distribution settings page or from the Connection Function associated distributions table. Navigate to your distribution settings, scroll to the **Viewer mutual authentication (mTLS)** section, select your Connection Function, and save the changes.

Advanced revocation scenarios

For more complex certificate revocation requirements, consider these additional configurations:

Topics

- Convert Certificate Revocation Lists (CRL) to KeyValueStore format

- Handle multiple Certificate Authorities
- Add custom data to connection logs
- Manage CRL updates
- Plan KeyValueStore capacity

Convert Certificate Revocation Lists (CRL) to KeyValueStore format

If you have a Certificate Revocation List (CRL) file, you can convert it to KeyValueStore JSON format using OpenSSL and jq:

Convert CRL to KeyValueStore format

Extract serial numbers from the CRL file:

```
openssl crl -text -noout -in rfc5280_CRL.crl | \
  awk '/Serial Number:/ {print $3}' | \
  cut -d'=' -f2 | \
  sed 's/../&:/g;s/:$//' >> serialnumbers.txt
```

Convert the serial numbers to KeyValueStore JSON format:

```
jq -R -s 'split("\n") | map(select(length > 0)) | {data: map({"key": ., "value": ""})}'
 \
  serialnumbers.txt >> serialnumbers_kvs.json
```

Upload the formatted file to S3 and create the KeyValueStore as described in Step 1.

Handle multiple Certificate Authorities

When your TrustStore contains multiple Certificate Authorities (CAs), include the issuer information in your KeyValueStore keys to avoid conflicts between certificates from different CAs that might have the same serial number.

For multi-CA scenarios, use a combination of the issuer's SHA1 hash and the serial number as the key:

```
import cf from 'cloudfront';

async function connectionHandler(connection) {
```

```
    const kvsHandle = cf.kvs();
    const clientCert = connection.clientCertInfo;

    // Create composite key with issuer hash and serial number
    const issuer = clientCert.issuer.replace(/[^a-zA-Z0-9]/g, '').substring(0, 20);
    const serialno = clientCert.serialNumber;
    const compositeKey = `${issuer}_${serialno}`;

    const cert_revoked = await kvsHandle.exists(compositeKey);

    if (cert_revoked) {
        console.log(`Blocking revoked cert: ${serialno} from issuer: ${issuer}`);
        connection.deny();
    } else {
        connection.allow();
    }
}
```

> **Note**
>
> Using issuer identifier + serial number creates longer keys, which might reduce the total number of entries you can store in the KeyValueStore.

Add custom data to connection logs

Connection functions can add custom data to CloudFront connection logs using the logCustomData method. With this, you can include revocation check results, certificate information, or other relevant data in your logs.

```
async function connectionHandler(connection) {
    const kvsHandle = cf.kvs();
    const clientSerialNumber = connection.clientCertInfo.serialNumber;
    const serialNumberExistsInKvs = await kvsHandle.exists(clientSerialNumber);

    if (serialNumberExistsInKvs) {
        // Log revocation details to connection logs
        connection.logCustomData(`REVOKED:${clientSerialNumber}:DENIED`);
        console.log("Connection denied - certificate revoked");
        return connection.deny();
    }
```

```
    // Log successful validation
    connection.logCustomData(`VALID:${clientSerialNumber}:ALLOWED`);
    console.log("Connection allowed");
    return connection.allow();
}
```

Custom data is limited to 800 bytes of valid UTF-8 text. If you exceed this limit, CloudFront truncates the data to the nearest valid UTF-8 boundary.

> **Note**
>
> Custom data logging only works when connection logs are enabled for your distribution. If connection logs are not configured, the logCustomData method is a no-op.

Manage CRL updates

Certificate Authorities can issue two types of CRLs:

- **Full CRLs**: Contain a complete list of all revoked certificates
- **Delta CRLs**: Only list certificates revoked since the last full CRL

For full CRL updates, create a new KeyValueStore with the updated data and redirect the Connection Function association to the new KeyValueStore. This approach is simpler than calculating differences and performing incremental updates.

For delta CRL updates, use the update-keys command to add new revoked certificates to the existing KeyValueStore:

```
aws cloudfront update-key-value-store \
  --name "revoked-serials-kvs" \
  --if-match "current-etag" \
  --put file://delta-revoked-serials.json
```

Plan KeyValueStore capacity

KeyValueStore has a 5 MB size limit and supports up to 10 million key-value pairs. Plan your revocation list capacity based on your key format and data size:

- **Serial number only**: Efficient storage for simple revocation checking

- **Issuer identifier + serial number**: Longer keys for multi-CA environments

For large revocation lists, consider implementing a tiered approach where you maintain separate KeyValueStores for different certificate categories or time periods.

Customize at the edge with Lambda@Edge

Lambda@Edge is an extension of AWS Lambda. Lambda@Edge is a compute service that you can use to execute functions that customize the content that Amazon CloudFront delivers. You can author Node.js or Python functions in the Lambda console in one AWS Region, US East (N. Virginia).

After you create the function, you can add triggers using the Lambda console or CloudFront console so that the functions run in AWS locations that are closer to the viewer, without provisioning or managing servers. Optionally, you can use Lambda and CloudFront API operations to set up your functions and triggers programmatically.

Lambda@Edge scales automatically, from a few requests per day to thousands per second. Processing requests at AWS locations closer to the viewer instead of on origin servers significantly reduces latency and improves the user experience.

> **Note**
>
> Lambda@Edge isn't supported with gRPC requests. For more information, see Using gRPC with CloudFront distributions.

Topics

- How Lambda@Edge works with requests and responses
- Ways to use Lambda@Edge
- Get started with Lambda@Edge functions (console)
- Set up IAM permissions and roles for Lambda@Edge
- Write and create a Lambda@Edge function
- Add triggers for a Lambda@Edge function
- Test and debug Lambda@Edge functions
- Delete Lambda@Edge functions and replicas
- Lambda@Edge event structure

- Work with requests and responses
- Lambda@Edge example functions

How Lambda@Edge works with requests and responses

When you associate a CloudFront distribution with a Lambda@Edge function, CloudFront intercepts requests and responses at CloudFront edge locations. You can execute Lambda functions when the following CloudFront events occur:

- When CloudFront receives a request from a viewer (viewer request)
- Before CloudFront forwards a request to the origin (origin request)
- When CloudFront receives a response from the origin (origin response)
- Before CloudFront returns the response to the viewer (viewer response)

If you're using AWS WAF, the Lambda@Edge viewer request is executed after any AWS WAF rules are applied.

For more information, see Work with requests and responses and Lambda@Edge event structure.

Ways to use Lambda@Edge

There are many uses for Lambda@Edge processing with your Amazon CloudFront distribution, such as the following examples:

- A Lambda function can inspect cookies and rewrite URLs so that users see different versions of a site for A/B testing.
- CloudFront can return different objects to viewers based on the device they're using by checking the `User-Agent` header, which includes information about the devices. For example, CloudFront can return different images based on the screen size of their device. Similarly, the function could consider the value of the `Referer` header and cause CloudFront to return the images to bots that have the lowest available resolution.
- Or you could check cookies for other criteria. For example, on a retail website that sells clothing, if you use cookies to indicate which color a user chose for a jacket, a Lambda function can change the request so that CloudFront returns the image of a jacket in the selected color.
- A Lambda function can generate HTTP responses when CloudFront viewer request or origin request events occur.

- A function can inspect headers or authorization tokens, and insert a header to control access to your content before CloudFront forwards the request to your origin.
- A Lambda function can also make network calls to external resources to confirm user credentials, or fetch additional content to customize a response.

For more information, including example code, see Lambda@Edge example functions.

For more information about setting up Lambda@Edge in the console, see Tutorial: Create a basic Lambda@Edge function (console).

Get started with Lambda@Edge functions (console)

With Lambda@Edge, you can use CloudFront triggers to invoke a Lambda function. When you associate a CloudFront distribution with a Lambda function, CloudFront intercepts requests and responses at CloudFront edge locations and runs the function. Lambda functions can improve security or customize information close to your viewers to improve performance.

The following list provides a basic overview of how to create and use Lambda functions with CloudFront.

Overview: Creating and using Lambda functions with CloudFront

1. Create a Lambda function in the US East (N. Virginia) Region.
2. Save and publish a numbered version of the function.

   If you want to change the function, you must edit the $LATEST version of the function in the US East (N. Virginia) Region. Then, before you set it up to work with CloudFront, you publish a new numbered version.
3. Associate the function with a CloudFront distribution and cache behavior. Then specify one or more CloudFront events (*triggers*) that cause the function to execute. For example, you can create a trigger for the function to execute when CloudFront receives a request from a viewer.
4. When you create a trigger, Lambda creates replicas of the function at AWS locations around the world.

> **Tip**
>
> For more information, see creating and updating functions, the event structure, and adding CloudFront triggers. You can also find more ideas and get code samples in Lambda@Edge example functions.

For a step-by-step tutorial, see the following topic:

Topics

- Tutorial: Create a basic Lambda@Edge function (console)

Tutorial: Create a basic Lambda@Edge function (console)

This tutorial shows you how to get started with Lambda@Edge by creating and configuring an example Node.js function that runs in CloudFront. This example adds HTTP security headers to a response when CloudFront retrieves a file. (This can improve security and privacy for a website.)

You don't need your own website for this tutorial. However, when you choose to create your own Lambda@Edge solution, you follow similar steps and select from the same options.

Topics

- Step 1: Sign up for an AWS account
- Step 2: Create a CloudFront distribution
- Step 3: Create your function
- Step 4: Add a CloudFront trigger to run the function
- Step 5: Verify that the function runs
- Step 6: Troubleshoot issues
- Step 7: Clean up your example resources
- Related information

Step 1: Sign up for an AWS account

If you haven't already done so, sign up for an AWS account. For more information, see Sign up for an AWS account.

Step 2: Create a CloudFront distribution

Before you create the example Lambda@Edge function, you must have a CloudFront environment to work with that includes an origin to serve content from.

For this example, you create a CloudFront distribution that uses an Amazon S3 bucket as the origin for the distribution. If you already have an environment to use, you can skip this step.

To create a CloudFront distribution with an Amazon S3 origin

1. Create an Amazon S3 bucket with a file or two, such as image files, for sample content. For help, follow the steps in Upload your content to Amazon S3. Make sure that you set permissions to grant public read access to the objects in your bucket.
2. Create a CloudFront distribution and add your S3 bucket as an origin, by following the steps in Create a CloudFront web distribution. If you already have a distribution, you can add the bucket as an origin for that distribution instead.

> **Tip**
>
> Make a note of your distribution ID. Later in this tutorial when you add a CloudFront trigger for your function, you must choose the ID for your distribution in a dropdown list—for example, `E653W22221KDDL`.

Step 3: Create your function

In this step, you create a Lambda function from a blueprint template in the Lambda console. The function adds code to update security headers in your CloudFront distribution.

To create a Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.

> **Important**
>
> Make sure that you're in the **US-East-1 (N. Virginia)** AWS Region (**us-east-1**). You must be in this Region to create Lambda@Edge functions.

2. Choose **Create function**.

3. On the **Create function** page, choose **Use a blueprint**, and then filter for the CloudFront blueprints by entering **`cloudfront`** in the search field.

   > **Note**
   >
   > CloudFront blueprints are available only in the **US-East-1 (N. Virginia)** Region (**us-east-1**).

4. Choose the **Modify HTTP response header** blueprint as the template for your function.
5. Enter the following information about your function:

   - **Function name** – Enter a name for your function.
   - **Execution role** – Choose how to set the permissions for your function. To use the recommended basic Lambda@Edge permissions policy template, choose **Create a new role from AWS policy templates**.
   - **Role name** – Enter a name for the role that the policy template creates.
   - **Policy templates** – Lambda automatically adds the policy template **Basic Lambda@Edge permissions** because you chose a CloudFront blueprint as the basis for your function. This policy template adds execution role permissions that allow CloudFront to run your Lambda function for you in CloudFront locations around the world. For more information, see Set up IAM permissions and roles for Lambda@Edge.
6. Choose **Create function** at the bottom of the page.
7. In the **Deploy to Lambda@Edge** pane that appears, choose **Cancel**. (For this tutorial, you must modify the function code before deploying the function to Lambda@Edge.)
8. Scroll down to the **Code source** section of the page.
9. Replace the template code with a function that modifies the security headers that your origin returns. For example, you could use code similar to the following:

```
'use strict';
export const handler = (event, context, callback) => {

    //Get contents of response
    const response = event.Records[0].cf.response;
    const headers = response.headers;

    //Set new headers
```

```
    headers['strict-transport-security'] = [{key: 'Strict-Transport-Security',
 value: 'max-age=63072000; includeSubdomains; preload'}];
    headers['content-security-policy'] = [{key: 'Content-Security-Policy', value:
 "default-src 'none'; img-src 'self'; script-src 'self'; style-src 'self'; object-
src 'none'"}];
    headers['x-content-type-options'] = [{key: 'X-Content-Type-Options', value:
 'nosniff'}];
    headers['x-frame-options'] = [{key: 'X-Frame-Options', value: 'DENY'}];
    headers['x-xss-protection'] = [{key: 'X-XSS-Protection', value: '1;
 mode=block'}];
    headers['referrer-policy'] = [{key: 'Referrer-Policy', value: 'same-origin'}];

    //Return modified response
    callback(null, response);
};
```

10. Choose **File**, **Save** to save your updated code.
11. Choose **Deploy**.

Proceed to the next section to add a CloudFront trigger to run the function.

Step 4: Add a CloudFront trigger to run the function

Now that you have a Lambda function to update security headers, configure the CloudFront trigger to run your function to add the headers in any response that CloudFront receives from the origin for your distribution.

To configure the CloudFront trigger for your function

1. In the Lambda console, on the **Function overview** page for your function, choose **Add trigger**.
2. For **Trigger configuration**, choose **CloudFront**.
3. Choose **Deploy to Lambda@Edge**.
4. In the **Deploy to Lambda@Edge** pane, under **Configure CloudFront trigger**, enter the following information:

   - **Distribution** – The CloudFront distribution ID to associate with your function. In the dropdown list, choose the distribution ID.
   - **Cache behavior** – The cache behavior to use with the trigger. For this example, leave the value set to *, which means your distribution's default cache behavior. For more information, see Cache behavior settings in the All distribution settings reference topic.

- **CloudFront event** – The trigger that specifies when your function runs. The security headers function should run whenever CloudFront returns a response from the origin. In the dropdown list, choose **Origin response**. For more information, see Add triggers for a Lambda@Edge function.

5. Select the **Confirm deploy to Lambda@Edge** check box.
6. Choose **Deploy** to add the trigger and replicate the function to AWS locations worldwide.
7. Wait for the function to replicate. This typically takes several minutes.

   You can check to see if replication is finished by going to the CloudFront console and viewing your distribution. Wait for the distribution status to change from **Deploying** to a date and time, which means that your function has been replicated. To verify that the function works, follow the steps in the next section.

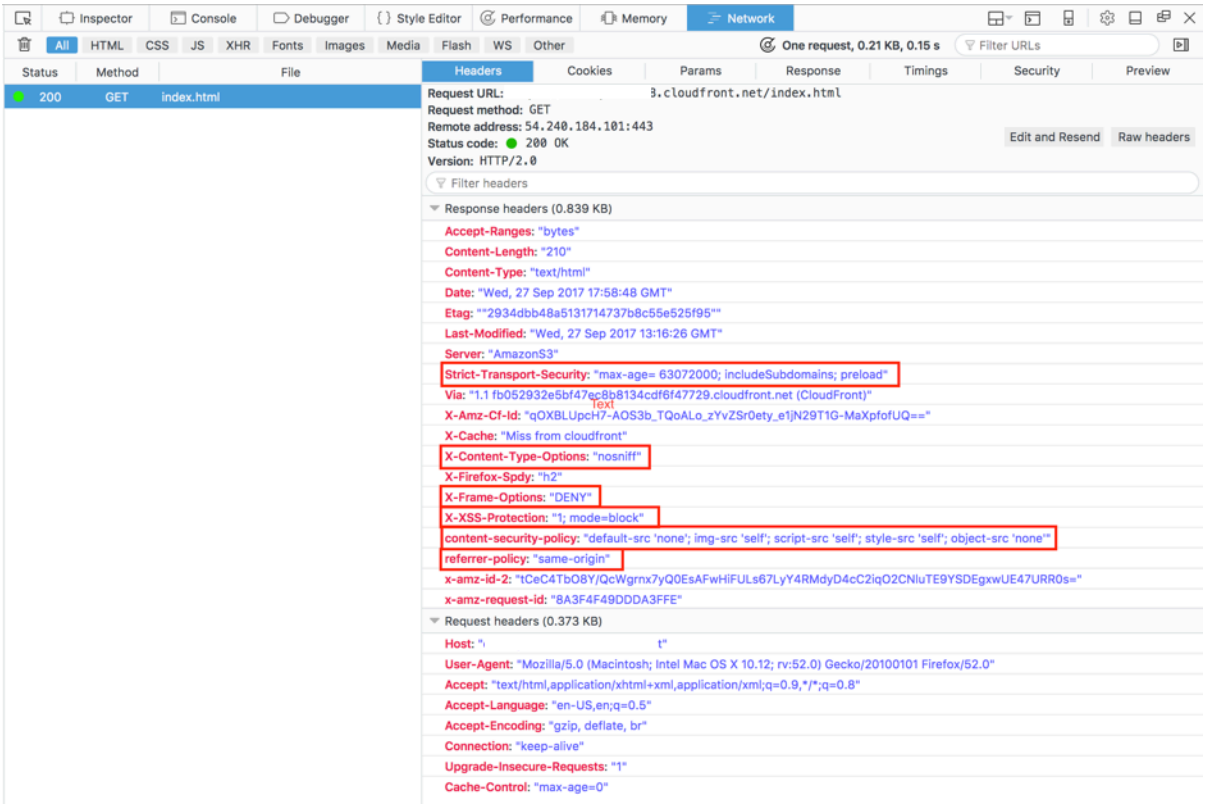
Step 5: Verify that the function runs

Now that you've created your Lambda function and configured a trigger to run it for a CloudFront distribution, check to make sure that the function is accomplishing what you expect it to. In this example, you check the HTTP headers that CloudFront returns, to make sure that the security headers are added.

To verify that your Lambda@Edge function adds security headers

1. In a browser, enter the URL for a file in your S3 bucket. For example, you might use a URL similar to `https://d111111abcdef8.cloudfront.net/image.jpg`.

   For more information about the CloudFront domain name to use in the file URL, see Customize the URL format for files in CloudFront.
2. Open your browser's Web Developer toolbar. For example, in your browser window in Chrome, open the context menu, and then choose **Inspect**.
3. Choose the **Network** tab.
4. Reload the page to view your image, and then choose an HTTP request on the left pane. You see the HTTP headers displayed in a separate pane.
5. Look through the list of HTTP headers to verify that the expected security headers are included in the list. For example, you might see headers similar to those shown in the following screenshot.



If the security headers are included in your headers list, great! You've successfully created your first Lambda@Edge function. If CloudFront returns errors or there are other issues, continue to the next step to troubleshoot the issues.

Step 6: Troubleshoot issues

If CloudFront returns errors or doesn't add the security headers as expected, you can investigate your function's execution by looking at CloudWatch Logs. Be sure to use the logs stored in the AWS location that is closest to the location where the function is executed.

For example, if you view the file from London, try changing the Region in the CloudWatch console to Europe (London).

To examine CloudWatch logs for your Lambda@Edge function

1. Sign in to the AWS Management Console and open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. Change **Region** to the location that is shown when you view the file in your browser. This is where the function is executing.
3. In the left pane, choose **Logs** to view the logs for your distribution.

For more information, see Monitor CloudFront metrics with Amazon CloudWatch.

Step 7: Clean up your example resources

If you created an Amazon S3 bucket and CloudFront distribution just for this tutorial, delete the AWS resources that you allocated so that you no longer accrue charges. After you delete your AWS resources, any content that you added is no longer available.

Tasks

- Delete the S3 bucket
- Delete the Lambda function
- Delete the CloudFront distribution

Delete the S3 bucket

Before you delete your Amazon S3 bucket, make sure that logging is disabled for the bucket. Otherwise, AWS continues to write logs to your bucket as you delete it.

To disable logging for a bucket

1. Open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. Select your bucket, and then choose **Properties**.
3. From **Properties**, choose **Logging**.
4. Clear the **Enabled** check box.
5. Choose **Save**.

Now, you can delete your bucket. For more information, see Deleting a bucket in the *Amazon Simple Storage Service Console User Guide*.

Delete the Lambda function

For instructions to delete the Lambda function association and optionally the function itself, see Delete Lambda@Edge functions and replicas.

Delete the CloudFront distribution

Before you delete a CloudFront distribution, you must disable it. A disabled distribution is no longer functional and does not accrue charges. You can enable a disabled distribution at any time. After you delete a disabled distribution, it's no longer available.

To disable and delete a CloudFront distribution

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Select the distribution that you want to disable, and then choose **Disable**.
3. When prompted for confirmation, choose **Yes, Disable**.
4. Select the disabled distribution, and then choose **Delete**.
5. When prompted for confirmation, choose **Yes, Delete**.

Related information

Now that you have a basic idea of how Lambda@Edge functions work, learn more by reading the following:

- Lambda@Edge example functions
- Lambda@Edge Design Best Practices
- Reducing Latency and Shifting Compute to the Edge with Lambda@Edge

Set up IAM permissions and roles for Lambda@Edge

To configure Lambda@Edge, you must have the following IAM permissions and roles for AWS Lambda:

- IAM permissions – These permissions allow you to create your Lambda function and associate it with your CloudFront distribution.
- A Lambda function execution role (IAM role) – The Lambda service principals assume this role to execute your function.
- Service-linked roles for Lambda@Edge – The service-linked roles allow specific AWS services to replicate Lambda functions to AWS Regions and to enable CloudWatch to use CloudFront log files.

IAM permissions required to associate Lambda@Edge functions with CloudFront distributions

In addition to the IAM permissions that you need for Lambda, you need the following permissions to associate Lambda functions with CloudFront distributions:

- `lambda:GetFunction` – Grants permission to get configuration information for your Lambda function and a presigned URL to download a `.zip` file that contains the function.
- `lambda:EnableReplication*` – Grants permission to the resource policy so that the Lambda replication service can get the function code and configuration.
- `lambda:DisableReplication*` – Grants permission to the resource policy so that the Lambda replication service can delete the function.

  > **Important**
  >
  > You must add the asterisk (*) at the end of the `lambda:EnableReplication*` and `lambda:DisableReplication*` actions.

- For the resource, specify the ARN of the function version that you want to execute when a CloudFront event occurs, such as the following example:

  `arn:aws:lambda:us-east-1:123456789012:function:TestFunction:2`

- `iam:CreateServiceLinkedRole` – Grants permission to create a service-linked role that Lambda@Edge uses to replicate Lambda functions in CloudFront. After you configure Lambda@Edge for the first time, the service-linked role is automatically created for you. You don't need to add this permission to other distributions that use Lambda@Edge.

- `cloudfront:UpdateDistribution` or `cloudfront:CreateDistribution` – Grants permission to update or create a distribution.

For more information, see the following topics:

- Identity and Access Management for Amazon CloudFront
- Lambda resource access permissions in the *AWS Lambda Developer Guide*

Function execution role for service principals

You must create an IAM role that the `lambda.amazonaws.com` and `edgelambda.amazonaws.com` service principals can assume when they execute your function.

> **Tip**
>
> When you create your function in the Lambda console, you can choose to create a new execution role by using an AWS policy template. This step *automatically* adds the required Lambda@Edge permissions to execute your function. See Step 5 in the Tutorial: Creating a simple Lambda@Edge function.

For more information about creating an IAM role manually, see Creating roles and attaching policies (console) in the *IAM User Guide*.

Example: Role trust policy

You can add this role under the **Trust Relationship** tab in the IAM console. Don't add this policy under the **Permissions** tab.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": [
               "lambda.amazonaws.com",
               "edgelambda.amazonaws.com"
            ]
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

For more information about the permissions that you need to grant to the execution role, see Lambda resource access permissions in the *AWS Lambda Developer Guide*.

Notes

- By default, whenever a CloudFront event triggers a Lambda function, data is written to CloudWatch Logs. If you want to use these logs, the execution role needs permission to write data to CloudWatch Logs. You can use the predefined AWSLambdaBasicExecutionRole to grant permission to the execution role.

  For more information about CloudWatch Logs, see the section called "Edge function logs".
- If your Lambda function code accesses other AWS resources, such as reading an object from an S3 bucket, the execution role needs permission to perform that action.

Service-linked roles for Lambda@Edge

Lambda@Edge uses IAM service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to a service. Service-linked roles are predefined by the service and include all of the permissions that the service requires to call other AWS services on your behalf.

Lambda@Edge uses the following IAM service-linked roles:

- **AWSServiceRoleForLambdaReplicator** – Lambda@Edge uses this role to allow Lambda@Edge to replicate functions to AWS Regions.

  When you first add a Lambda@Edge trigger in CloudFront, a role named AWSServiceRoleForLambdaReplicator is created automatically to allow Lambda@Edge to replicate functions to AWS Regions. This role is required to use Lambda@Edge functions. The ARN for the AWSServiceRoleForLambdaReplicator role looks like the following example:

  ```
  arn:aws:iam::123456789012:role/aws-service-role/
  replicator.lambda.amazonaws.com/AWSServiceRoleForLambdaReplicator
  ```
- **AWSServiceRoleForCloudFrontLogger** – CloudFront uses this role to push log files into CloudWatch. You can use log files to debug Lambda@Edge validation errors.

The AWSServiceRoleForCloudFrontLogger role is created automatically when you add a Lambda@Edge function association to allow CloudFront to push Lambda@Edge error log files to CloudWatch. The ARN for the AWSServiceRoleForCloudFrontLogger role looks like this:

```
arn:aws:iam::account_number:role/aws-service-role/
logger.cloudfront.amazonaws.com/AWSServiceRoleForCloudFrontLogger
```

A service-linked role makes setting up and using Lambda@Edge easier because you don't have to manually add the necessary permissions. Lambda@Edge defines the permissions of its service-linked roles, and only Lambda@Edge can assume the roles. The defined permissions include the trust policy and the permissions policy. You can't attach the permissions policy to any other IAM entity.

You must remove any associated CloudFront or Lambda@Edge resources before you can delete a service-linked role. This helps protect your Lambda@Edge resources so that you don't remove a service-linked role that is still required to access active resources.

For more information about service-linked roles, see Service-linked roles for CloudFront.

Service-linked role permissions for Lambda@Edge

Lambda@Edge uses two service-linked roles, named **AWSServiceRoleForLambdaReplicator** and **AWSServiceRoleForCloudFrontLogger**. The following sections describe the permissions for each of these roles.

Contents

- Service-linked role permissions for Lambda replicator
- Service-linked role permissions for CloudFront logger

Service-linked role permissions for Lambda replicator

This service-linked role allows Lambda to replicate Lambda@Edge functions to AWS Regions.

The AWSServiceRoleForLambdaReplicator service-linked role trusts the `replicator.lambda.amazonaws.com` service to assume the role.

The role permissions policy allows Lambda@Edge to complete the following actions on the specified resources:

- `lambda:CreateFunction` on `arn:aws:lambda:*:*:function:*`
- `lambda:DeleteFunction` on `arn:aws:lambda:*:*:function:*`
- `lambda:DisableReplication` on `arn:aws:lambda:*:*:function:*`
- `iam:PassRole` on `all AWS resources`
- `cloudfront:ListDistributionsByLambdaFunction` on `all AWS resources`

Service-linked role permissions for CloudFront logger

This service-linked role allows CloudFront to push log files into CloudWatch so that you can debug Lambda@Edge validation errors.

The AWSServiceRoleForCloudFrontLogger service-linked role trusts the `logger.cloudfront.amazonaws.com` service to assume the role.

The role permissions policy allows Lambda@Edge to complete the following actions on the specified `arn:aws:logs:*:*:log-group:/aws/cloudfront/*` resource:

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

You must configure permissions to allow an IAM entity (such as a user, group, or role) to delete the Lambda@Edge service-linked roles. For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating service-linked roles for Lambda@Edge

You don't typically manually create the service-linked roles for Lambda@Edge. The service creates the roles for you automatically in the following scenarios:

- When you first create a trigger, the service creates the AWSServiceRoleForLambdaReplicator role (if it doesn't already exist). This role allows Lambda to replicate Lambda@Edge functions to AWS Regions.

  If you delete the service-linked role, the role will be created again when you add a new trigger for Lambda@Edge in a distribution.

- When you update or create a CloudFront distribution that has a Lambda@Edge association, the service creates the AWSServiceRoleForCloudFrontLogger role (if the role doesn't already exist). This role allows CloudFront to push your log files to CloudWatch.

  If you delete the service-linked role, the role will be created again when you update or create a CloudFront distribution that has a Lambda@Edge association.

To manually create these service-linked roles, you can run the following AWS Command Line Interface (AWS CLI) commands:

To create the AWSServiceRoleForLambdaReplicator role

- Run the following command.

```
aws iam create-service-linked-role --aws-service-name
 replicator.lambda.amazonaws.com
```

To create the AWSServiceRoleForCloudFrontLogger role

- Run the following command.

```
aws iam create-service-linked-role --aws-service-name
 logger.cloudfront.amazonaws.com
```

Editing Lambda@Edge service-linked roles

Lambda@Edge doesn't allow you to edit the AWSServiceRoleForLambdaReplicator or AWSServiceRoleForCloudFrontLogger service-linked roles. After the service has created a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can use IAM to edit the role description. For more information, see Editing a service-linked role in the *IAM User Guide*.

Supported AWS Regions for Lambda@Edge service-linked roles

CloudFront supports using service-linked roles for Lambda@Edge in the following AWS Regions:

- US East (N. Virginia) – `us-east-1`
- US East (Ohio) – `us-east-2`

- US West (N. California) – `us-west-1`
- US West (Oregon) – `us-west-2`
- Asia Pacific (Mumbai) – `ap-south-1`
- Asia Pacific (Seoul) – `ap-northeast-2`
- Asia Pacific (Singapore) – `ap-southeast-1`
- Asia Pacific (Sydney) – `ap-southeast-2`
- Asia Pacific (Tokyo) – `ap-northeast-1`
- Europe (Frankfurt) – `eu-central-1`
- Europe (Ireland) – `eu-west-1`
- Europe (London) – `eu-west-2`
- South America (São Paulo) – `sa-east-1`

Write and create a Lambda@Edge function

To use Lambda@Edge, you *write* the code for your AWS Lambda function. To help you write Lambda@Edge functions, see the following resources:

- Lambda@Edge event structure – Understand the event structure to use with Lambda@Edge.
- Lambda@Edge example functions – Example functions, such as A/B testing and generating an HTTP redirect.

The programming model for using Node.js or Python with Lambda@Edge is the same as using Lambda in an AWS Region. For more information, see Building Lambda functions with Node.js or Building Lambda functions with Python in the *AWS Lambda Developer Guide*.

In your Lambda@Edge function, include the `callback` parameter and return the applicable object for request or response events:

- **Request events** – Include the `cf.request` object in the response.

  If you're generating a response, include the `cf.response` object in the response. For more information, see Generate HTTP responses in request triggers.
- **Response events** – Include the `cf.response` object in the response.

After you write your own code or use one of the examples, you then create the function in Lambda. To create a function or edit an existing one, see the following topics:

Topics

- Create a Lambda@Edge function
- Edit a Lambda function

After you create the function in Lambda, you set up Lambda to run the function based on specific CloudFront events, which are called *triggers*. For more information, see Add triggers for a Lambda@Edge function.

Create a Lambda@Edge function

To set up AWS Lambda to run Lambda functions that are based on CloudFront events, follow this procedure.

To create a Lambda@Edge function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. If you already have one or more Lambda functions, choose **Create function**.

   If you don't have any functions, choose **Get Started Now**.
3. In the Region list at the top of the page, choose **US East (N. Virginia)**.
4. Create a function using your own code or create a function starting with a CloudFront blueprint.

   - To create a function using your own code, choose **Author from scratch**.
   - To display a list of blueprints for CloudFront, enter **cloudfront** in the filter field, and then choose **Enter**.

     If you find a blueprint that you want to use, choose the name of the blueprint.
5. In the **Basic information** section, specify the following values:

   a. **Name** – Enter a name for your function.

   b. **Role** – To get started quickly, choose **Create new role from template(s)**. You can also choose **Choose an existing role** or **Create a custom role**, and then follow the prompts to complete the information for this section.

c. **Role name** – Enter a name for the role.

d. **Policy templates** – Choose **Basic Edge Lambda permissions**.

6. If you chose **Author from scratch** in step 4, skip to step 7.

   If you chose a blueprint in step 4, you can use the **cloudfront** section to create one trigger, which associates this function with a cache in a CloudFront distribution and a CloudFront event. We recommend that you choose **Remove** at this point, so there isn't a trigger for the function when it's created. Then you can add triggers later.

   > **Tip**
   >
   > We recommend that you test and debug the function before adding triggers. If you add a trigger now, the function will run as soon as you create the function and it finishes replicating to AWS locations around the world, and the corresponding distribution is deployed.

7. Choose **Create function**.

   Lambda creates two versions of your function: $LATEST and Version 1. You can edit only the $LATEST version, but the console initially displays Version 1.

8. To edit the function, choose **Version 1** near the top of the page, under the ARN for the function. Then, on the **Versions** tab, choose **$LATEST**. (If you left the function and then returned to it, the button label is **Qualifiers**.)

9. On the **Configuration** tab, choose the applicable **Code entry type**. Then follow the prompts to edit or upload your code.

10. For **Runtime**, choose the value based on your function's code.

11. In the **Tags** section, add any applicable tags.

12. Choose **Actions**, and then choose **Publish new version**.

13. Enter a description for the new version of the function.

14. Choose **Publish**.

15. Test and debug the function. For more information about testing in the Lambda console, see Invoke a Lambda function using the console in the *AWS Lambda Developer Guide*.

16. When you're ready to have the function execute for CloudFront events, publish another version and edit the function to add triggers. For more information, see Add triggers for a Lambda@Edge function.

Edit a Lambda function

After you create a Lambda@Edge function, you can use the Lambda console to edit it.

Notes

- The original version is labeled $LATEST.
- You can edit only the $LATEST version.
- Each time you edit the $LATEST version, you must publish a new numbered version.
- You can't create triggers for $LATEST.
- When you publish a new version of a function, Lambda doesn't automatically copy triggers from the previous version to the new version. You must reproduce the triggers for the new version.
- When you add a trigger for a CloudFront event to a function, if there's already a trigger for the same distribution, cache behavior, and event for an earlier version of the same function, Lambda deletes the trigger from the earlier version.
- After you make updates to a CloudFront distribution, like adding triggers, you must wait for the changes to propagate to edge locations before the functions you've specified in the triggers will work.

To edit a Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. In the Region list at the top of the page, choose **US East (N. Virginia)**.
3. In the list of functions, choose the name of the function.

   By default, the console displays the $LATEST version. You can view earlier versions (choose **Qualifiers**), but you can only edit $LATEST.
4. On the **Code** tab, for **Code entry type**, choose to edit the code in the browser, upload a .zip file, or upload a file from Amazon S3.
5. Choose either **Save** or **Save and test**.
6. Choose **Actions**, and choose **Publish new version**.

7. In the **Publish new version from $LATEST** dialog box, enter a description of the new version. This description appears in the list of versions, along with an automatically generated version number.
8. Choose **Publish**.

   The new version automatically becomes the latest version. The version number appears on the **Version** in the upper-left corner of the page.

   > **Note**
   >
   > If you haven't added triggers for your function yet, see Add triggers for a Lambda@Edge function.

9. Choose the **Triggers** tab.
10. Choose **Add trigger**.
11. In the **Add trigger** dialog box, choose the dotted box, and then choose **CloudFront**.

    > **Note**
    >
    > If you've already created one or more triggers for a function, CloudFront is the default service.

12. Specify the following values to indicate when you want the Lambda function to execute.
    a. **Distribution ID** – Choose the ID of the distribution that you want to add the trigger to.
    b. **Cache behavior** – Choose the cache behavior that specifies the objects that you want to execute the function on.
    c. **CloudFront event** – Choose the CloudFront event that causes the function to execute.
    d. **Enable trigger and replicate** – Select this check box so Lambda replicates the function to AWS Regions globally.
13. Choose **Submit**.
14. To add more triggers for this function, repeat steps 10 through 13.

For more information about testing and debugging the function in the Lambda console, see Invoke a Lambda function using the console in the *AWS Lambda Developer Guide*.

When you're ready to have the function execute for CloudFront events, publish another version and edit the function to add triggers. For more information, see Add triggers for a Lambda@Edge function.

Add triggers for a Lambda@Edge function

A Lambda@Edge trigger is one combination of a CloudFront distribution, cache behavior, and event that causes a function to execute. For example, you can create a trigger that causes the function to execute when CloudFront receives a request from a viewer for a specific cache behavior you set up for your distribution. You can specify one or more CloudFront triggers.

> **Tip**
>
> When you create a CloudFront distribution, you specify settings that tell CloudFront how to respond when it receives different requests. The default settings are called the *default cache behavior* for the distribution. You can set up additional cache behaviors that define how CloudFront responds under specific circumstances, for example, when it receives a request for a specific file type. For more information, see Cache behavior settings.

When you first create a Lambda function, you can specify only *one* trigger. You can add more triggers to the same function later by using the Lambda console or by editing the distribution in the CloudFront console.

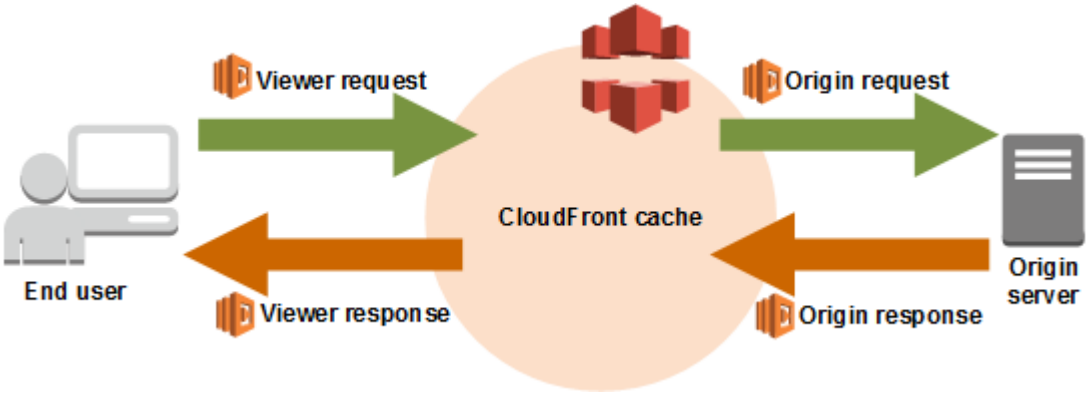
- The Lambda console works well if you want to add more triggers to a function for the same CloudFront distribution.
- The CloudFront console can be better if you want to add triggers for multiple distributions because it's easier to find the distribution that you want to update. You can also update other CloudFront settings at the same time.

Topics

- CloudFront events that can trigger a Lambda@Edge function
- Choose the event to trigger the function
- Add triggers to a Lambda@Edge function

CloudFront events that can trigger a Lambda@Edge function

For each cache behavior in an Amazon CloudFront distribution, you can add up to four triggers (associations) that cause a Lambda function to execute when specific CloudFront events occur. CloudFront triggers can be based on one of four CloudFront events, as shown in the following diagram.



The CloudFront events that can be used to trigger Lambda@Edge functions are the following:

Viewer request

The function executes when CloudFront receives a request from a viewer, before it checks to see whether the requested object is in the CloudFront cache.

The function doesn't execute in the following cases:

- When fetching a custom error page.
- When CloudFront automatically redirects an HTTP request to HTTPS (when the value of the Viewer protocol policy is **Redirect HTTP to HTTPS**).

Origin request

The function executes *only* when CloudFront forwards a request to your origin. When the requested object is in the CloudFront cache, the function doesn't execute.

Origin response

The function executes after CloudFront receives a response from the origin and before it caches the object in the response. Note that the function executes even if an error is returned from the origin.

The function doesn't execute in the following cases:

- When the requested file is in the CloudFront cache and is not expired.
- When the response is generated from a function that was triggered by an origin request event.

Viewer response

The function executes before returning the requested file to the viewer. Note that the function executes regardless of whether the file is already in the CloudFront cache.

The function doesn't execute in the following cases:

- When the origin returns an HTTP status code of 400 or higher.
- When a custom error page is returned.
- When the response is generated from a function that was triggered by a viewer request event.
- When CloudFront automatically redirects an HTTP request to HTTPS (when the value of the Viewer protocol policy is **Redirect HTTP to HTTPS**).

When you add multiple triggers to the same cache behavior, you can use them to run the same function or run different functions for each trigger. You can also associate the same function with more than one distribution.

> **Note**
>
> When a CloudFront event triggers the execution of a Lambda function, the function must finish *before* CloudFront can continue.
> For example, if a Lambda function is triggered by a CloudFront viewer request event, CloudFront won't return a response to the viewer or forward the request to the origin until the Lambda function finishes running.
> This means that each request that triggers a Lambda function increases latency for the request, so you want the function to execute as fast as possible.

Choose the event to trigger the function

When you're deciding which CloudFront event you want to use to trigger a Lambda function, consider the following:

I want CloudFront to cache objects that are changed by a Lambda function

To cache an object that was modified by a Lambda function so that CloudFront can serve the object from the edge location the next time it's requested, use the *origin request* or *origin response* event.

This reduces the load on the origin, reduces latency for subsequent requests, and reduces the cost of invoking Lambda@Edge on subsequent requests.

For example, if you want to add, remove, or change headers for objects that are returned by the origin and you want CloudFront to cache the result, use the origin response event.

I want the function to execute for every request

To execute the function for every request that CloudFront receives for the distribution, use the *viewer request* or *viewer response* events.

Origin request and origin response events occur only when a requested object isn't cached in an edge location and CloudFront forwards a request to the origin.

I want the function to change the cache key

To change a value that you're using as a basis for caching, use the *viewer request* event.

For example, if a function changes the URL to include a language abbreviation in the path (for example, because the user chose their language from a dropdown list), use the viewer request event:

- **URL in the viewer request** – https://example.com/en/index.html
- **URL when the request comes from an IP address in Germany** – https://example.com/de/index.html

You also use the viewer request event if you're caching based on cookies or request headers.

> **Note**
>
> If the function changes cookies or headers, configure CloudFront to forward the applicable part of the request to the origin. For more information, see the following topics:
>
> - Cache content based on cookies
> - Cache content based on request headers

The function affects the response from the origin

To change the request in a way that affects the response from the origin, use the *origin request* event.

Typically, most viewer request events aren't forwarded to the origin. CloudFront responds to a request with an object that's already in the edge cache. If the function changes the request based on an origin request event, CloudFront caches the response to the changed origin request.

Add triggers to a Lambda@Edge function

You can use the AWS Lambda console or Amazon CloudFront console to add a trigger to your Lambda@Edge function.

> **Important**
>
> You can create triggers only for numbered versions of your function (not the **$LATEST**).

Lambda console

To add triggers for CloudFront events to a Lambda@Edge function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. In the Region list at the top of the page, choose **US East (N. Virginia)**.
3. On the **Functions** page, choose the name of the function that you want to add triggers for.
4. On the **Function overview** page, choose the **Versions** tab.
5. Choose the version that you want to add triggers to.

   After you choose a version, the name of the button changes to **Version: $LATEST** or **Version:** *version number*.
6. Choose the **Triggers** tab.
7. Choose **Add trigger**.
8. For **Trigger configuration**, choose **Select a source**, enter **`cloudfront`**, and then choose **CloudFront**.

> **Note**
>
> If you've already created one or more triggers, CloudFront is the default service.

9. Specify the following values to indicate when you want the Lambda function to execute.

   a. **Distribution** – Choose the distribution that you want to add the trigger to.

   b. **Cache behavior** – Choose the cache behavior that specifies the objects that you want to execute the function on.

   > **Note**
   >
   > If you specify * for the cache behavior, the Lambda function deploys to the default cache behavior.

   c. **CloudFront event** – Choose the CloudFront event that causes the function to execute.

   d. **Include body** – Select this check box if you want to access the request body in your function.

   e. **Confirm deploy to Lambda@Edge** – Select this check box so that AWS Lambda replicates the function to AWS Regions globally.

10. Choose **Add**.

    The function starts to process requests for the specified CloudFront events when the updated CloudFront distribution is deployed. To determine whether a distribution is deployed, choose **Distributions** in the navigation pane. When a distribution is deployed, the value of the **Status** column for the distribution changes from **Deploying** to the date and time of deployment.

CloudFront console

To add triggers for CloudFront events to a Lambda@Edge function

1. Get the ARN of the Lambda function that you want to add triggers for:

   a. Sign in to the AWS Management Console and open the AWS Lambda console at https://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.

   b. In the list of Regions at the top of the page, choose **US East (N. Virginia)**.

c. In the list of functions, choose the name of the function that you want to add triggers to.

d. On the **Function overview** page, choose the **Versions** tab, and choose the numbered version that you want to add triggers to.

e. Choose the **Copy ARN** button to copy the ARN to your clipboard. The ARN for the Lambda function looks something like this:

```
arn:aws:lambda:us-east-1:123456789012:function:TestFunction:2
```

The number at the end (**2** in this example) is the version number of the function.

2. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
3. In the list of distributions, choose the ID of the distribution that you want to add triggers to.
4. Choose the **Behaviors** tab.
5. Select the cache behavior that you want to add triggers to, and then choose **Edit**.
6. For **Function associations**, in the **Function type** list, choose **Lambda@Edge** for when you want the function to execute: for viewer requests, viewer responses, origin requests, or origin responses.

   For more information, see Choose the event to trigger the function.
7. In the **Function ARN / Name** text box, paste the ARN of the Lambda function that you want to execute when the chosen event occurs. This is the value that you copied from the Lambda console.
8. Select **Include body** if you want to access the request body in your function.

   If you just want to replace the request body, you don't need to select this option.
9. To execute the same function for more event types, repeat steps 6 and 7.
10. Choose **Save changes**.
11. To add triggers to more cache behaviors for this distribution, repeat steps 5 through 10.

    The function starts to process requests for the specified CloudFront events when the updated CloudFront distribution is deployed. To determine whether a distribution is deployed, choose **Distributions** in the navigation pane. When a distribution is deployed, the value of the **Status** column for the distribution changes from **Deploying** to the time and date of deployment.

Test and debug Lambda@Edge functions

It's important to test your Lambda@Edge function code standalone, to make sure that it completes the intended task, and to do integration testing, to make sure that the function works correctly with CloudFront.

During integration testing or after your function has been deployed, you might need to debug CloudFront errors, such as HTTP 5xx errors. Errors can be an invalid response returned from the Lambda function, execution errors when the function is triggered, or errors due to execution throttling by the Lambda service. Sections in this topic share strategies for determining which type of failure is the issue, and then steps you can take to correct the problem.

> **Note**
>
> When you review CloudWatch log files or metrics when you're troubleshooting errors, be aware that they are displayed or stored in the AWS Region closest to the location where the function executed. So, if you have a website or web application with users in the United Kingdom, and you have a Lambda function associated with your distribution, for example, you must change the Region to view the CloudWatch metrics or log files for the London AWS Region. For more information, see the section called “Determine the Lambda@Edge Region”.

Topics

- Test your Lambda@Edge functions
- Identify Lambda@Edge function errors in CloudFront
- Troubleshoot invalid Lambda@Edge function responses (validation errors)
- Troubleshoot Lambda@Edge function execution errors
- Determine the Lambda@Edge Region
- Determine if your account pushes logs to CloudWatch

Test your Lambda@Edge functions

There are two steps to testing your Lambda function: standalone testing and integration testing.

Test standalone functionality

Before you add your Lambda function to CloudFront, make sure to test the functionality first by using the testing capabilities in the Lambda console or by using other methods. For more information about testing in the Lambda console, see Invoke a Lambda function using the console in the *AWS Lambda Developer Guide*.

Test your function's operation in CloudFront

It's important to complete integration testing, where your function is associated with a distribution and runs based on a CloudFront event. Make sure that the function is triggered for the right event, and returns a response that is valid and correct for CloudFront. For example, make sure that the event structure is correct, that only valid headers are included, and so on.

As you iterate on integration testing with your function in the Lambda console, refer to the steps in the Lambda@Edge tutorial as you modify your code or change the CloudFront trigger that calls your function. For example, make sure that you're working in a numbered version of your function, as described in this step of the tutorial: Step 4: Add a CloudFront trigger to run the function.

As you make changes and deploy them, be aware that it will take several minutes for your updated function and CloudFront triggers to replicate across all Regions. This typically takes a few minutes but can take up to 15 minutes.

You can check to see if replication is finished by going to the CloudFront console and viewing your distribution.

To check if your replication has finished deploying

1. Open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the distribution name.
3. Check for the distribution status to change from **In Progress** back to **Deployed**, which means that your function has been replicated. Then follow the steps in the next section to verify that the function works.

Be aware that testing in the console only validates your function's logic, and does not apply any service quotas (formerly known as limits) that are specific to Lambda@Edge.

Identify Lambda@Edge function errors in CloudFront

After you've verified that your function logic works correctly, you might still see HTTP 5xx errors when your function runs in CloudFront. HTTP 5xx errors can be returned for a variety of reasons, which can include Lambda function errors or other issues in CloudFront.

- If you use Lambda@Edge functions, you can use graphs in the CloudFront console to help track down what's causing the error, and then work to fix it. For example, you can see if HTTP 5xx errors are caused by CloudFront or by Lambda functions, and then, for specific functions, you can view related log files to investigate the issue.
- To troubleshoot HTTP errors in general in CloudFront, see the troubleshooting steps in the following topic: Troubleshooting error response status codes in CloudFront.

What causes Lambda@Edge function errors in CloudFront

There are several reasons why a Lambda function might cause an HTTP 5xx error, and the troubleshooting steps you should take depend on the type of error. Errors can be categorized as the following:

A Lambda function execution error

An execution error results when CloudFront doesn't get a response from Lambda because there are unhandled exceptions in the function or there's an error in the code. For example, if the code includes callback(Error).

An invalid Lambda function response is returned to CloudFront

After the function runs, CloudFront receives a response from Lambda. An error is returned if the object structure of the response doesn't conform to the Lambda@Edge event structure, or the response contains invalid headers or other invalid fields.

The execution in CloudFront is throttled due to Lambda service quotas (formerly known as limits)

The Lambda service throttles executions in each Region, and returns an error if you exceed the quota. For more information, see Quotas on Lambda@Edge.

How to determine the type of failure

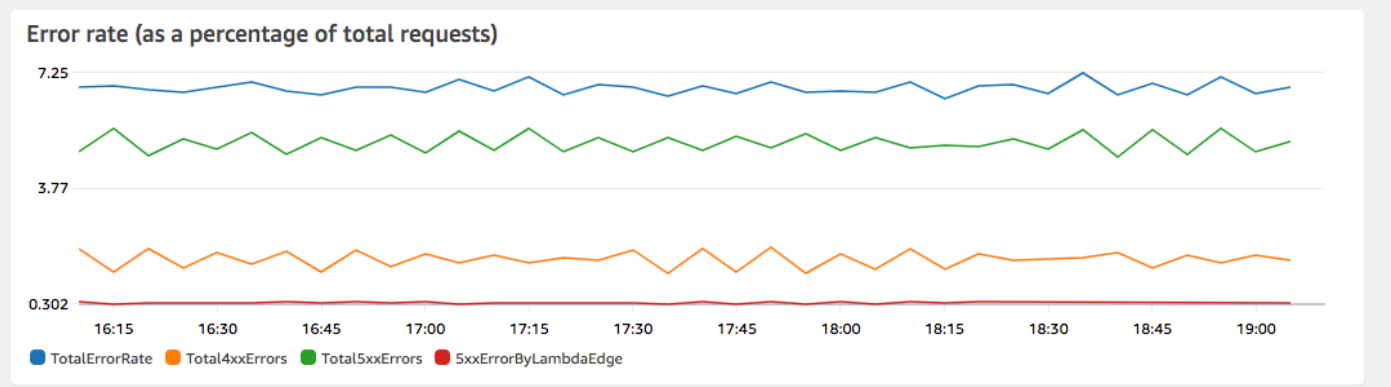
To help you decide where to focus as you debug and work to resolve errors returned by CloudFront, it's helpful to identify why CloudFront is returning an HTTP error. To get started, you can use the

graphs provided in the **Monitoring** section of the CloudFront console on the AWS Management Console. For more information about viewing graphs in the **Monitoring** section of the CloudFront console, see Monitor CloudFront metrics with Amazon CloudWatch.

The following graphs can be especially helpful when you want to track down whether errors are being returned by origins or by a Lambda function, and to narrow down the type of issue when it's an error from a Lambda function.

Error rates graph

One of the graphs that you can view on the **Overview** tab for each of your distributions is an **Error rates** graph. This graph displays the rate of errors as a percentage of total requests coming to your distribution. The graph shows the total error rate, total 4xx errors, total 5xx errors, and total 5xx errors from Lambda functions. Based on the error type and volume, you can take steps to investigate and troubleshoot the cause.



- If you see Lambda errors, you can investigate further by looking at the specific types of errors that the function returns. The **Lambda@Edge errors** tab includes graphs that categorize function errors by type to help you pinpoint the issue for a specific function.
- If you see CloudFront errors, you can troubleshoot and work to fix origin errors or change your CloudFront configuration. For more information, see Troubleshooting error response status codes in CloudFront.

Execution errors and invalid function responses graphs

The **Lambda@Edge errors** tab includes graphs that categorize the Lambda@Edge errors for a specific distribution, by type. For example, one graph shows all execution errors by AWS Region.

To make it easier to troubleshoot issues, you can look for specific problems by opening and examining the log files for specific functions by Region.

To view log files for a specific function by Region

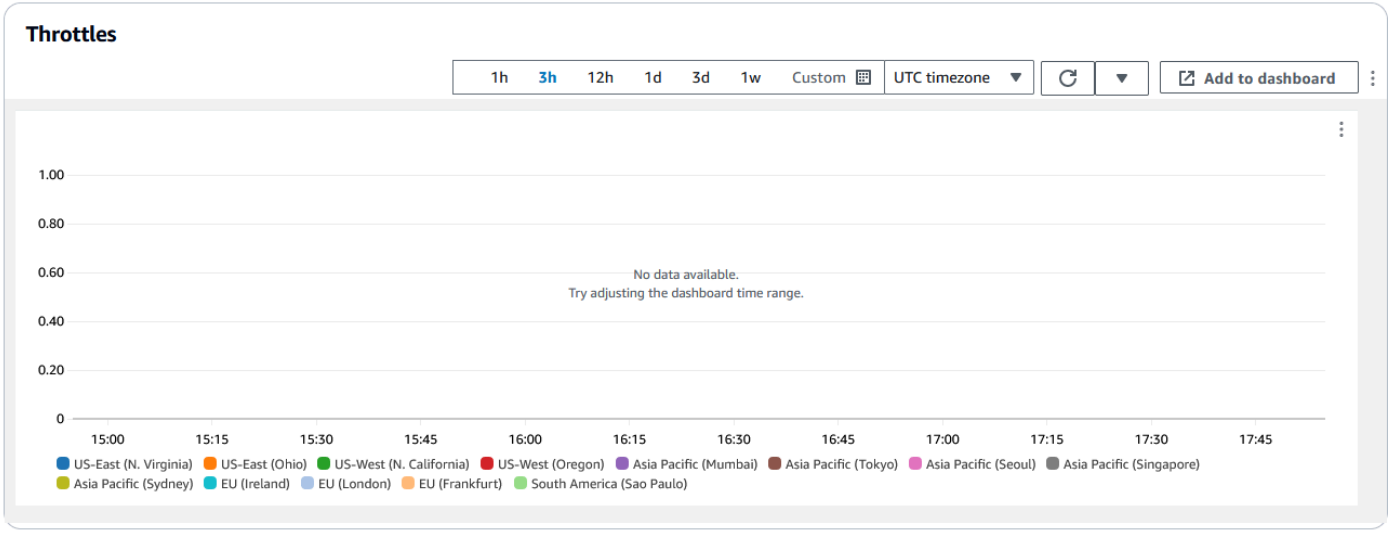
1. On the **Lambda@Edge errors** tab, under **Associated Lambda@Edge functions**, choose the function name, and then choose **View metrics**.
2. Next, on the page with your function name, in the upper-right corner, choose **View function logs**, and then choose a Region.

   For example, if you see issues in the **Errors** graph for the US West (Oregon) Region, choose that Region from the dropdown list. This opens the Amazon CloudWatch console.
3. In the CloudWatch console for that Region, under **Log streams**, choose a log stream to view the events for the function.

In addition, read the following sections in this chapter for more recommendations about troubleshooting and fixing errors.

Throttles graph

The **Lambda@Edge errors** tab also includes a **Throttles** graph. On occasion, the Lambda service throttles your function invocations on per Region basis, if you reach the regional concurrency quota (formerly known as limit). If you see a limit exceeded error, your function has reached a quota that the Lambda service imposes on executions in a Region. For more information, including how to request an increase in the quota, see Quotas on Lambda@Edge.



For an example about how to use this information in troubleshooting HTTP errors, see Four steps for debugging your content delivery on AWS.

Troubleshoot invalid Lambda@Edge function responses (validation errors)

If you identify that your problem is a Lambda validation error, it means that your Lambda function is returning an invalid response to CloudFront. Follow the guidance in this section to take steps to review your function and make sure that your response conforms to CloudFront requirements.

CloudFront validates the response from a Lambda function in two ways:

- **The Lambda response must conform to the required object structure.** Examples of bad object structure include the following: unparsable JSON, missing required fields, and an invalid object in the response. For more information, see the Lambda@Edge event structure.
- **The response must include only valid object values.** An error will occur if the response includes a valid object but has values that are not supported. Examples include the following: adding or updating disallowed or read-only headers (see Restrictions on edge functions), exceeding the maximum body size (see *Restrictions on the Size of the Generated Response* in the Lambda@Edge Errors topic), and invalid characters or values (see the Lambda@Edge event structure).

When Lambda returns an invalid response to CloudFront, error messages are written to log files which CloudFront pushes to CloudWatch in the Region of where the Lambda function executed. It's the default behavior to send the log files to CloudWatch when there's an invalid response. However, if you associated a Lambda function with CloudFront before the functionality was released, it might not be enabled for your function. For more information, see *Determine if Your Account Pushes Logs to CloudWatch* later in the topic.

CloudFront pushes log files to the Region corresponding to where your function executed, in the log group that's associated with your distribution. Log groups have the following format: `/aws/cloudfront/LambdaEdge/DistributionId`, where `DistributionId` is your distribution's ID. To determine the Region where you can find the CloudWatch log files, see *Determining the Lambda@Edge Region* later in this topic.

If the error is reproducible, you can create a new request that results in the error and then find the request id in a failed CloudFront response (`X-Amz-Cf-Id` header) to locate a single failure in log files. The log file entry includes information that can help you identify why the error is being returned, and also lists the corresponding Lambda request id so you can analyze the root cause in the context of a single request.

If an error is intermittent, you can use CloudFront access logs to find the request id for a request that has failed, and then search CloudWatch logs for the corresponding error messages. For more information, see the previous section, *Determining the Type of Failure*.

Troubleshoot Lambda@Edge function execution errors

If the problem is a Lambda execution error, it can be helpful to create logging statements for Lambda functions, to write messages to CloudWatch log files that monitor the execution of your function in CloudFront and determine if it's working as expected. Then you can search for those statements in the CloudWatch log files to verify that your function is working.

> **Note**
>
> Even if you haven't changed your Lambda@Edge function, updates to the Lambda function execution environment might affect it and could return an execution error. For information about testing and migrating to a later version, see Upcoming updates to the AWS Lambda and AWS Lambda@Edge execution environment.

Determine the Lambda@Edge Region

To see the Regions where your Lambda@Edge function is receiving traffic, view metrics for the function on the CloudFront console on the AWS Management Console. Metrics are displayed for each AWS Region. On the same page, you can choose a Region and view log files for that Region so you can investigate issues. You must review CloudWatch log files in the correct AWS Region to see the log files created when CloudFront executed your Lambda function.

For more information about viewing graphs in the **Monitoring** section of the CloudFront console, see Monitor CloudFront metrics with Amazon CloudWatch.

Determine if your account pushes logs to CloudWatch

By default, CloudFront enables logging invalid Lambda function responses, and pushes the log files to CloudWatch by using one of the Service-linked roles for Lambda@Edge. If you have Lambda@Edge functions that you added to CloudFront before the invalid Lambda function response log feature was released, logging is enabled when you next update your Lambda@Edge configuration, for example, by adding a CloudFront trigger.

You can verify that pushing the log files to CloudWatch is enabled for your account by doing the following:

- **Check to see if the logs appear in CloudWatch** – Make sure that you look in the Region where the Lambda@Edge function executed. For more information, see Determine the Lambda@Edge Region.
- **Determine if the related service-linked role exists in your account in IAM –** You must have the IAM role `AWSServiceRoleForCloudFrontLogger` in your account. For more information about this role, see Service-linked roles for Lambda@Edge.

Delete Lambda@Edge functions and replicas

You can delete a Lambda@Edge function only when the replicas of the function have been deleted by CloudFront. Replicas of a Lambda function are automatically deleted in the following situations:

- After you remove the last association for the function from all of your CloudFront distributions. If more than one distribution uses a function, the replicas are deleted only after you remove the function association from the last distribution.
- After you delete the last distribution that a function was associated with.

Replicas are typically deleted within a few hours. You cannot manually delete Lambda@Edge function replicas. This helps prevent a situation where a replica is deleted that is still in use, which would result in an error.

> **Warning**
>
> Don't build applications that use Lambda@Edge function replicas outside of CloudFront. These replicas are deleted when their associations with distributions are removed, or when distributions themselves are deleted. The replica that an outside application depends on might be removed without warning, causing it to fail.

To delete a Lambda@Edge function association from a CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the ID of the distribution with the Lambda@Edge function association that you want to delete.
3. Choose the **Behaviors** tab.

4. Select the cache behavior that has the Lambda@Edge function association that you want to delete, and then choose **Edit**.
5. Under **Function associations**, **Function type**, choose **No association** to delete the Lambda@Edge function association.
6. Choose **Save changes**.

After you delete a Lambda@Edge function association from a CloudFront distribution, you can optionally delete the Lambda function or function version from AWS Lambda. Wait a few hours after deleting the function association so that the Lambda@Edge function replicas can be cleaned up. After that, you can delete the function by using the Lambda console, AWS CLI, Lambda API, or an AWS SDK.

You can also delete a specific *version* of a Lambda function if the version doesn't have any CloudFront distributions associated with it. After removing all the associations for a Lambda function version, wait a few hours. Then you can delete the function version.

Lambda@Edge event structure

The following topics describe the request and response event objects that CloudFront passes to a Lambda@Edge function when it's triggered.

Topics

- Dynamic origin selection
- Request events
- Response events

Dynamic origin selection

You can use the path pattern in a cache behavior to route requests to an origin based on the path and name of the requested object, such as `images/*.jpg`. Using Lambda@Edge, you can also route requests to an origin based on other characteristics, such as the values in request headers.

There are a number of ways that this dynamic origin selection can be useful. For example, you can distribute requests across origins in different geographic areas to help with global load balancing. Or you can selectively route requests to different origins that each serve a particular function: bot handling, SEO optimization, authentication, and so on. For code examples that demonstrate how to use this feature, see Content-based dynamic origin selection - examples.

In the CloudFront origin request event, the `origin` object in the event structure contains information about the origin that the request would be routed to, based on the path pattern. You can update the values in the `origin` object to route a request to a different origin. When you update the `origin` object, you don't need to define the origin in the distribution. You can also replace an Amazon S3 origin object with a custom origin object, and vice versa. You can only specify a single origin per request, though; either a custom origin or an Amazon S3 origin, but not both.

Request events

The following topics show the structure of the object that CloudFront passes to a Lambda function for viewer and origin request events. These examples show a `GET` request with no body. Following the examples is a list of all the possible fields in viewer and origin request events.

Topics

- Example viewer request
- Example origin request
- Request event fields

Example viewer request

The following example shows a viewer request event object.

```
{
  "Records": [
    {
      "cf": {
        "config": {
          "distributionDomainName": "d111111abcdef8.cloudfront.net",
          "distributionId": "EDFDVBD6EXAMPLE",
          "eventType": "viewer-request",
          "requestId": "4TyzHTaYWb1GX1qTfsHhEqV6HUDd_BzoBZnwfnvQc_1oF26ClkoUSEQ=="
        },
        "request": {
          "clientIp": "203.0.113.178",
          "headers": {
            "host": [
              {
                "key": "Host",
                "value": "d111111abcdef8.cloudfront.net"
```

```
            }
          ],
          "user-agent": [
            {
              "key": "User-Agent",
              "value": "curl/7.66.0"
            }
          ],
          "accept": [
            {
              "key": "accept",
              "value": "*/*"
            }
          ]
        },
        "method": "GET",
        "querystring": "",
        "uri": "/"
      }
    }
  ]
}
```

Example origin request

The following example shows an origin request event object.

```
{
  "Records": [
    {
      "cf": {
        "config": {
          "distributionDomainName": "d111111abcdef8.cloudfront.net",
          "distributionId": "EDFDVBD6EXAMPLE",
          "eventType": "origin-request",
          "requestId": "4TyzHTaYWb1GX1qTfsHhEqV6HUDd_BzoBZnwfnvQc_1oF26ClkoUSEQ=="
        },
        "request": {
          "clientIp": "203.0.113.178",
          "headers": {
            "x-forwarded-for": [
              {
                "key": "X-Forwarded-For",
```

```
                "value": "203.0.113.178"
              }
            ],
            "user-agent": [
              {
                "key": "User-Agent",
                "value": "Amazon CloudFront"
              }
            ],
            "via": [
              {
                "key": "Via",
                "value": "2.0 2afae0d44e2540f472c0635ab62c232b.cloudfront.net
(CloudFront)"
              }
            ],
            "host": [
              {
                "key": "Host",
                "value": "example.org"
              }
            ],
            "cache-control": [
              {
                "key": "Cache-Control",
                "value": "no-cache"
              }
            ]
          },
          "method": "GET",
          "origin": {
            "custom": {
              "customHeaders": {},
              "domainName": "example.org",
              "keepaliveTimeout": 5,
              "path": "",
              "port": 443,
              "protocol": "https",
              "readTimeout": 30,
              "responseCompletionTimeout": 30,
              "sslProtocols": [
                "TLSv1",
                "TLSv1.1",
                "TLSv1.2"
```

```
              ]
            }
          },
          "querystring": "",
          "uri": "/"
        }
      }
    }
  ]
}
```

Request event fields

Request event object data is contained in two subobjects: `config` (`Records.cf.config`) and `request` (`Records.cf.request`). The following lists describe each subobject's fields.

Fields in the config object

The following list describes the fields in the `config` object (`Records.cf.config`).

`distributionDomainName` (read-only)

The domain name of the distribution that's associated with the request.

`distributionId` (read-only)

The ID of the distribution that's associated with the request.

`eventType` (read-only)

The type of trigger that's associated with the request: `viewer-request` or `origin-request`.

`requestId` (read-only)

An encrypted string that uniquely identifies a viewer-to-CloudFront request. The `requestId` value also appears in CloudFront access logs as `x-edge-request-id`. For more information, see Access logs (standard logs) and Log file fields.

Fields in the request object

The following list describes the fields in the `request` object (`Records.cf.request`).

`clientIp` (read-only)

The IP address of the viewer that made the request. If the viewer used an HTTP proxy or a load balancer to send the request, the value is the IP address of the proxy or load balancer.

headers (read/write)

The headers in the request. Note the following:

- The keys in the `headers` object are lowercase versions of standard HTTP header names. Using lowercase keys gives you case-insensitive access to the header values.
- Each header object (for example, `headers["accept"]` or `headers["host"]`) is an array of key–value pairs. For a given header, the array contains one key–value pair for each value in the request.
- `key` contains the case-sensitive name of the header as it appeared in the HTTP request; for example, `Host`, `User-Agent`, `X-Forwarded-For`, `Cookie`, and so on.
- `value` contains the header value as it appeared in the HTTP request.
- When your Lambda function adds or modifies request headers and you don't include the header `key` field, Lambda@Edge automatically inserts a header `key` using the header name that you provide. Regardless of how you've formatted the header name, the header key that's inserted automatically is formatted with initial capitalization for each part, separated by hyphens (-).

  For example, you can add a header like the following, without a header `key`:

  ```
  "user-agent": [
    {
      "value": "ExampleCustomUserAgent/1.X.0"
    }
  ]
  ```

  In this example, Lambda@Edge automatically inserts `"key": "User-Agent"`.

For information about restrictions on header usage, see Restrictions on edge functions.

`method` (read-only)

The HTTP method of the request.

`querystring` (read/write)

The query string, if any, in the request. If the request doesn't include a query string, the event object still includes `querystring` with an empty value. For more information about query strings, see Cache content based on query string parameters.

`uri` (read/write)

The relative path of the requested object. If your Lambda function modifies the `uri` value, note the following:

- The new `uri` value must begin with a forward slash (/).
- When a function changes the `uri` value, that changes the object that the viewer is requesting.
- When a function changes the `uri` value, that *doesn't* change the cache behavior for the request or the origin that the request is sent to.

`body` (read/write)

The body of the HTTP request. The `body` structure can contain the following fields:

`inputTruncated` (read-only)

A Boolean flag that indicates whether the body was truncated by Lambda@Edge. For more information, see Restrictions on the request body with the include body option.

`action` (read/write)

The action that you intend to take with the body. The options for `action` are the following:

- `read-only`: This is the default. When returning the response from the Lambda function, if `action` is read-only, Lambda@Edge ignores any changes to `encoding` or `data`.
- `replace`: Specify this when you want to replace the body sent to the origin.

`encoding` (read/write)

The encoding for the body. When Lambda@Edge exposes the body to the Lambda function, it first converts the body to base64-encoding. If you choose `replace` for the `action` to replace the body, you can opt to use `base64` (the default) or `text` encoding. If you specify `encoding` as `base64` but the body is not valid base64, CloudFront returns an error.

`data` (read/write)

The request body content.

`origin` (read/write) (origin events only)

The origin to send the request to. The `origin` structure must contain *exactly one origin*, which can be a custom origin or an Amazon S3 origin.

Depending on your origin type that you specify (custom or Amazon S3 origins), you must specify the following fields in your request:

`customHeaders` (read/write) (custom and Amazon S3 origins)

(Optional) You can include custom headers with the request by specifying a header name and value pair for each custom header. You can't add headers that are disallowed, and a header with the same name can't be present in `Records.cf.request.headers`. The notes about request headers also apply to custom headers. For more information, see Custom headers that CloudFront can't add to origin requests and Restrictions on edge functions.

`domainName` (read/write) (custom and Amazon S3 origins)

The domain name of the origin. The domain name can't be empty.

- **For custom origins** – Specify a DNS domain name, such as `www.example.com`. The domain name can't include a colon (:), and can't be an IP address. The domain name can be up to 253 characters.
- **For Amazon S3 origins** – Specify the DNS domain name of the Amazon S3 bucket, such as `amzn-s3-demo-bucket.s3.eu-west-1.amazonaws.com`. The name can be up to 128 characters, and must be all lowercase.

`path` (read/write) (custom and Amazon S3 origins)

The directory path at the origin where the request should locate content. The path should start with a forward slash (/) but shouldn't end with one (for example, it shouldn't end with `example-path/`). For custom origins only, the path should be URL encoded and have a maximum length of 255 characters.

`keepaliveTimeout` (read/write) (custom origins only)

How long, in seconds, that CloudFront should try to maintain the connection to the origin after receiving the last packet of the response. The value must be a number from 1–120, inclusive.

`port` (read/write) (custom origins only)

The port that CloudFront should connect to at your custom origin. The port must be 80, 443, or a number in the range of 1024–65535, inclusive.

`protocol` (read/write) (custom origins only)

The connection protocol that CloudFront should use when connecting to your origin. The value can be `http` or `https`.

`readTimeout` (read/write) (custom and Amazon S3 origins)

How long, in seconds, CloudFront should wait for a response after sending a request to your origin. This also specifies how long CloudFront should wait after receiving a packet of a response before receiving the next packet. The value must be a number from 1–120, inclusive.

If you need a higher quota, see Response timeout per origin.

`responseCompletionTimeout` (read/write) (custom and Amazon S3 origins)

The time (in seconds) that a request from CloudFront to the origin can stay open and wait for a response. If the complete response isn't received from the origin by this time, CloudFront ends the connection.

The value for `responseCompletionTimeout` must be equal to or greater than the value for the `readTimeout`. If you set this value to 0, it removes any previous value you set and returns to the default. You can also accomplish this by deleting the `responseCompletionTimeout` field from the event request.

`sslProtocols` (read/write) (custom origins only)

The minimum SSL/TLS protocol that CloudFront can use when establishing an HTTPS connection with your origin. Values can be any of the following: `TLSv1.2`, `TLSv1.1`, `TLSv1`, or `SSLv3`.

`authMethod` (read/write) (Amazon S3 origins only)

If you're using an origin access identity (OAI), set this field to `origin-access-identity`. If you aren't using an OAI, set it to none. If you set `authMethod` to `origin-access-identity`, there are several requirements:

- You must specify the `region` (see the following field).
- You must use the same OAI when you change the request from one Amazon S3 origin to another.

- You can't use an OAI when you change the request from a custom origin to an Amazon S3 origin.

> **Note**
>
> This field does not support origin access control (OAC).

`region` (read/write) (Amazon S3 origins only)

The AWS Region of your Amazon S3 bucket. This is required only when you set `authMethod` to `origin-access-identity`.

Response events

The following topics show the structure of the object that CloudFront passes to a Lambda function for viewer and origin response events. Following the examples is a list of all the possible fields in viewer and origin response events.

Topics

- Example origin response
- Example viewer response
- Response event fields

Example origin response

The following example shows an origin response event object.

```
{
  "Records": [
    {
      "cf": {
        "config": {
          "distributionDomainName": "d111111abcdef8.cloudfront.net",
          "distributionId": "EDFDVBD6EXAMPLE",
          "eventType": "origin-response",
          "requestId": "4TyzHTaYWb1GX1qTfsHhEqV6HUDd_BzoBZnwfnvQc_1oF26ClkoUSEQ=="
        },
        "request": {
```

```
          "clientIp": "203.0.113.178",
          "headers": {
            "x-forwarded-for": [
              {
                "key": "X-Forwarded-For",
                "value": "203.0.113.178"
              }
            ],
            "user-agent": [
              {
                "key": "User-Agent",
                "value": "Amazon CloudFront"
              }
            ],
            "via": [
              {
                "key": "Via",
                "value": "2.0 8f22423015641505b8c857a37450d6c0.cloudfront.net
(CloudFront)"
              }
            ],
            "host": [
              {
                "key": "Host",
                "value": "example.org"
              }
            ],
            "cache-control": [
              {
                "key": "Cache-Control",
                "value": "no-cache"
              }
            ]
          },
          "method": "GET",
          "origin": {
            "custom": {
              "customHeaders": {},
              "domainName": "example.org",
              "keepaliveTimeout": 5,
              "path": "",
              "port": 443,
              "protocol": "https",
              "readTimeout": 30,
```

```
            "responseCompletionTimeout": 30,
            "sslProtocols": [
              "TLSv1",
              "TLSv1.1",
              "TLSv1.2"
            ]
          }
        },
        "querystring": "",
        "uri": "/"
      },
      "response": {
        "headers": {
          "access-control-allow-credentials": [
            {
              "key": "Access-Control-Allow-Credentials",
              "value": "true"
            }
          ],
          "access-control-allow-origin": [
            {
              "key": "Access-Control-Allow-Origin",
              "value": "*"
            }
          ],
          "date": [
            {
              "key": "Date",
              "value": "Mon, 13 Jan 2020 20:12:38 GMT"
            }
          ],
          "referrer-policy": [
            {
              "key": "Referrer-Policy",
              "value": "no-referrer-when-downgrade"
            }
          ],
          "server": [
            {
              "key": "Server",
              "value": "ExampleCustomOriginServer"
            }
          ],
          "x-content-type-options": [
```

```
            {
              "key": "X-Content-Type-Options",
              "value": "nosniff"
            }
          ],
          "x-frame-options": [
            {
              "key": "X-Frame-Options",
              "value": "DENY"
            }
          ],
          "x-xss-protection": [
            {
              "key": "X-XSS-Protection",
              "value": "1; mode=block"
            }
          ],
          "content-type": [
            {
              "key": "Content-Type",
              "value": "text/html; charset=utf-8"
            }
          ],
          "content-length": [
            {
              "key": "Content-Length",
              "value": "9593"
            }
          ]
        },
        "status": "200",
        "statusDescription": "OK"
      }
    }
  }
]
}
```

Example viewer response

The following example shows a viewer response event object.

```
{
  "Records": [
```

```
    {
      "cf": {
        "config": {
          "distributionDomainName": "d111111abcdef8.cloudfront.net",
          "distributionId": "EDFDVBD6EXAMPLE",
          "eventType": "viewer-response",
          "requestId": "4TyzHTaYWb1GX1qTfsHhEqV6HUDd_BzoBZnwfnvQc_1oF26ClkoUSEQ=="
        },
        "request": {
          "clientIp": "203.0.113.178",
          "headers": {
            "host": [
              {
                "key": "Host",
                "value": "d111111abcdef8.cloudfront.net"
              }
            ],
            "user-agent": [
              {
                "key": "User-Agent",
                "value": "curl/7.66.0"
              }
            ],
            "accept": [
              {
                "key": "accept",
                "value": "*/*"
              }
            ]
          },
          "method": "GET",
          "querystring": "",
          "uri": "/"
        },
        "response": {
          "headers": {
            "access-control-allow-credentials": [
              {
                "key": "Access-Control-Allow-Credentials",
                "value": "true"
              }
            ],
            "access-control-allow-origin": [
              {
```

```
            "key": "Access-Control-Allow-Origin",
            "value": "*"
          }
        ],
        "date": [
          {
            "key": "Date",
            "value": "Mon, 13 Jan 2020 20:14:56 GMT"
          }
        ],
        "referrer-policy": [
          {
            "key": "Referrer-Policy",
            "value": "no-referrer-when-downgrade"
          }
        ],
        "server": [
          {
            "key": "Server",
            "value": "ExampleCustomOriginServer"
          }
        ],
        "x-content-type-options": [
          {
            "key": "X-Content-Type-Options",
            "value": "nosniff"
          }
        ],
        "x-frame-options": [
          {
            "key": "X-Frame-Options",
            "value": "DENY"
          }
        ],
        "x-xss-protection": [
          {
            "key": "X-XSS-Protection",
            "value": "1; mode=block"
          }
        ],
        "age": [
          {
            "key": "Age",
            "value": "2402"
```

```
            }
          ],
          "content-type": [
            {
              "key": "Content-Type",
              "value": "text/html; charset=utf-8"
            }
          ],
          "content-length": [
            {
              "key": "Content-Length",
              "value": "9593"
            }
          ]
        },
        "status": "200",
        "statusDescription": "OK"
      }
    }
  ]
}
```

Response event fields

Response event object data is contained in three subobjects: `config` (`Records.cf.config`), `request` (`Records.cf.request`), and `response` (`Records.cf.response`). For more information about the fields in the request object, see Fields in the request object. The following lists describe the fields in the `config` and `response` subobjects.

Fields in the config object

The following list describes the fields in the `config` object (`Records.cf.config`).

`distributionDomainName` (read-only)

The domain name of the distribution that's associated with the response.

`distributionId` (read-only)

The ID of the distribution that's associated with the response.

`eventType` (read-only)

The type of trigger that's associated with the response: `origin-response` or `viewer-response`.

`requestId` (read-only)

An encrypted string that uniquely identifies the viewer-to-CloudFront request that this response is associated with. The `requestId` value also appears in CloudFront access logs as `x-edge-request-id`. For more information, see Access logs (standard logs) and Log file fields.

Fields in the response object

The following list describes the fields in the `response` object (`Records.cf.response`). For information about using a Lambda@Edge function to generate an HTTP response, see Generate HTTP responses in request triggers.

`headers` (read/write)

The headers in the response. Note the following:

- The keys in the `headers` object are lowercase versions of standard HTTP header names. Using lowercase keys gives you case-insensitive access to the header values.
- Each header object (for example, `headers["content-type"]` or `headers["content-length"]`) is an array of key–value pairs. For a given header, the array contains one key–value pair for each value in the response.
- `key` contains the case-sensitive name of the header as it appears in the HTTP response; for example, `Content-Type`, `Content-Length`, `Cookie`, and so on.
- `value` contains the header value as it appears in the HTTP response.
- When your Lambda function adds or modifies response headers and you don't include the header `key` field, Lambda@Edge automatically inserts a header `key` using the header name that you provide. Regardless of how you've formatted the header name, the header key that's inserted automatically is formatted with initial capitalization for each part, separated by hyphens (-).

  For example, you can add a header like the following, without a header `key`:

```
"content-type": [
  {
```

```
    "value": "text/html;charset=UTF-8"
  }
]
```

In this example, Lambda@Edge automatically inserts `"key": "Content-Type"`.

For information about restrictions on header usage, see Restrictions on edge functions.

`status`

The HTTP status code of the response.

`statusDescription`

The HTTP status description of the response.

Work with requests and responses

To use Lambda@Edge requests and responses, see the following topics:

Topics

- Use Lambda@Edge functions with origin failover
- Generate HTTP responses in request triggers
- Update HTTP responses in origin response triggers
- Access the request body by choosing the include body option

Use Lambda@Edge functions with origin failover

You can use Lambda@Edge functions with CloudFront distributions that you've set up with origin groups, for example, for origin failover that you configure to help make sure high availability. To use a Lambda function with an origin group, specify the function in an origin request or origin response trigger for an origin group when you create the cache behavior.

For more information, see the following:

- **Create origin groups:** Create an origin group
- **How origin failover works with Lambda@Edge:** Use origin failover with Lambda@Edge functions

Generate HTTP responses in request triggers

When CloudFront receives a request, you can use a Lambda function to generate an HTTP response that CloudFront returns directly to the viewer without forwarding the response to the origin. Generating HTTP responses reduces the load on the origin, and typically also reduces latency for the viewer.

Some common scenarios for generating HTTP responses include the following:

- Returning a small webpage to the viewer
- Returning an HTTP 301 or 302 status code to redirect the user to another webpage
- Returning an HTTP 401 status code to the viewer when the user hasn't authenticated

A Lambda@Edge function can generate an HTTP response when the following CloudFront events occur:

Viewer request events

When a function is triggered by a viewer request event, CloudFront returns the response to the viewer and doesn't cache it.

Origin request events

When a function is triggered by an origin request event, CloudFront checks the edge cache for a response that was previously generated by the function.

- If the response is in the cache, the function isn't executed and CloudFront returns the cached response to the viewer.
- If the response isn't in the cache, the function is executed, CloudFront returns the response to the viewer, and also caches it.

To see some sample code for generating HTTP responses, see Lambda@Edge example functions. You can also replace the HTTP responses in response triggers. For more information, see Update HTTP responses in origin response triggers.

Programming model

This section describes the programming model for using Lambda@Edge to generate HTTP responses.

Topics

- Response object
- Errors
- Required fields

Response object

The response you return as the `result` parameter of the `callback` method should have the following structure (note that only the `status` field is required).

```
const response = {
    body: 'content',
    bodyEncoding: 'text' | 'base64',
    headers: {
        'header name in lowercase': [{
           key: 'header name in standard case',
           value: 'header value'
         }],
         ...
    },
    status: 'HTTP status code (string)',
    statusDescription: 'status description'
};
```

The response object can include the following values:

`body`

The body, if any, that you want CloudFront to return in the generated response.

`bodyEncoding`

The encoding for the value that you specified in the `body`. The only valid encodings are `text` and `base64`. If you include `body` in the `response` object but omit `bodyEncoding`, CloudFront treats the body as text.

If you specify `bodyEncoding` as `base64` but the body is not valid base64, CloudFront returns an error.

`headers`

Headers that you want CloudFront to return in the generated response. Note the following:

- The keys in the `headers` object are lowercase versions of standard HTTP header names. Using lowercase keys gives you case-insensitive access to the header values.
- Each header (for example, `headers["accept"]` or `headers["host"]`) is an array of key-value pairs. For a given header, the array contains one key-value pair for each value in the generated response.
- `key` (optional) is the case-sensitive name of the header as it appears in an HTTP request; for example, `accept` or `host`.
- Specify `value` as a header value.
- If you do not include the header key portion of the key-value pair, Lambda@Edge automatically inserts a header key using the header name that you provide. Regardless of how you've formatted the header name, the header key that is inserted is automatically formatted with initial capitalization for each part, separated by hyphens (-).

  For example, you can add a header like the following, without a header key: `'content-type': [{ value: 'text/html;charset=UTF-8' }]`

  In this example, Lambda@Edge creates the following header key: `Content-Type`.

For information about restrictions on header usage, see Restrictions on edge functions.

`status`

The HTTP status code. Provide the status code as a string. CloudFront uses the provided status code for the following:

- Return in the response
- Cache in the CloudFront edge cache, when the response was generated by a function that was triggered by an origin request event
- Log in CloudFront Access logs (standard logs)

If the `status` value isn't between 200 and 599, CloudFront returns an error to the viewer.

`statusDescription`

The description that you want CloudFront to return in the response, to accompany the HTTP status code. You don't need to use standard descriptions, such as OK for an HTTP status code of 200. Updating this field will only work if the client connects over HTTP/1.1. If the client connects over HTTP/2, any changes to the `statusDescription` will be ignored and the standard description will be returned to the client.

Errors

The following are possible errors for generated HTTP responses.

Response Contains a Body and Specifies 204 (No Content) for Status

When a function is triggered by a viewer request, CloudFront returns an HTTP 502 status code (Bad Gateway) to the viewer when both of the following are true:

- The value of `status` is 204 (No Content)
- The response includes a value for `body`

This is because Lambda@Edge imposes the optional restriction found in RFC 2616, which states that an `HTTP 204` response does not need to contain a message body.

Restrictions on the Size of the Generated Response

The maximum size of a response that is generated by a Lambda function depends on the event that triggered the function:

- **Viewer request events** – 40 KB
- **Origin request events** – 1 MB

If the response is larger than the allowed size, CloudFront returns an HTTP 502 status code (Bad Gateway) to the viewer.

Required fields

The `status` field is required.

All other fields are optional.

Update HTTP responses in origin response triggers

When CloudFront receives an HTTP response from the origin server, if there is an origin-response trigger associated with the cache behavior, you can modify the HTTP response to override what was returned from the origin.

Some common scenarios for updating HTTP responses include the following:

- Changing the status to set an HTTP 200 status code and creating static body content to return to the viewer when an origin returns an error status code (4xx or 5xx). For sample code, see Example: Use an origin response trigger to update the error status code to 200.

- Changing the status to set an HTTP 301 or HTTP 302 status code, to redirect the user to another website when an origin returns an error status code (4xx or 5xx). For sample code, see Example: Use an origin response trigger to update the error status code to 302.

> **Note**
>
> The function must return a status value between `200` and 599 (inclusive), otherwise CloudFront returns an error to the viewer.

You can also replace the HTTP responses in viewer and origin request events. For more information, see Generate HTTP responses in request triggers.

When you're working with the HTTP response, Lambda@Edge does not expose the body that is returned by the origin server to the origin-response trigger. You can generate a static content body by setting it to the desired value, or remove the body inside the function by setting the value to be empty. If you don't update the body field in your function, the original body returned by the origin server is returned back to viewer.

Access the request body by choosing the include body option

You can opt to have Lambda@Edge expose the body in a request for writable HTTP methods (POST, PUT, DELETE, and so on), so that you can access it in your Lambda function. You can choose read-only access, or you can specify that you'll replace the body.

To enable this option, choose **Include Body** when you create a CloudFront trigger for your function that's for a viewer request or origin request event. For more information, see Add triggers for a Lambda@Edge function, or to learn about using **Include Body** with your function, see Lambda@Edge event structure.

Scenarios when you might want to use this feature include the following:

- Processing web forms, like "contact us" forms, without sending customer input data back to origin servers.
- Gathering web beacon data that's sent by viewer browsers and processing it at the edge.

For sample code, see Lambda@Edge example functions.

> **Note**
>
> If the request body is large, Lambda@Edge truncates it. For detailed information about the maximum size and truncation, see Restrictions on the request body with the include body option.

Lambda@Edge example functions

See the following examples to use Lambda functions with Amazon CloudFront.

> **Note**
>
> If you choose runtime Node.js 18 or later for your Lambda@Edge function, an `index.mjs` file is created for you automatically. To use the following code examples, rename the `index.mjs` file to `index.js` instead.

Topics

- General examples
- Generate responses - examples
- Query strings - examples
- Personalize content by country or device type headers - examples
- Content-based dynamic origin selection - examples
- Update error statuses - examples
- Access the request body - examples

General examples

The following examples show common ways to use Lambda@Edge in CloudFront.

Topics

- Example: A/B testing
- Example: Override a response header

Example: A/B testing

You can use the following example to test two different versions of an image without creating redirects or changing the URL. This example reads the cookies in the viewer request and modifies the request URL accordingly. If the viewer doesn't send a cookie with one of the expected values, the example randomly assigns the viewer to one of the URLs.

Node.js

```
'use strict';

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    if (request.uri !== '/experiment-pixel.jpg') {
        // do not process if this is not an A-B test request
        callback(null, request);
        return;
    }

    const cookieExperimentA = 'X-Experiment-Name=A';
    const cookieExperimentB = 'X-Experiment-Name=B';
    const pathExperimentA = '/experiment-group/control-pixel.jpg';
    const pathExperimentB = '/experiment-group/treatment-pixel.jpg';

    /*
     * Lambda at the Edge headers are array objects.
     *
     * When a client sends multiple headers with the same name, each
     * header is a separate entry in the array. For example:
     * > GET /viewerRes/test HTTP/1.1
     * > X-Custom: valueA
     * > X-Custom: valueB
     *
     * You can access the first at headers["x-custom"][0].value
     * and the second at headers["x-custom"][1].value.
     *
     * CloudFront concatenates multiple Cookie headers into a single entry,
 separated by "; ".
     * For example:
     * > Cookie: First=1; Second=2
     * > Cookie: ClientCode=abc
```

```
     *
     * becomes headers["cookie"][0].value = "First=1; Second=2; ClientCode=abc"
     * This behavior is consistent across HTTP/1.1, HTTP/2, and HTTP/3.
     *
     * Header values are not parsed.
     */
    let experimentUri;
    if (headers.cookie) {
        for (let i = 0; i < headers.cookie.length; i++) {
            if (headers.cookie[i].value.indexOf(cookieExperimentA) >= 0) {
                console.log('Experiment A cookie found');
                experimentUri = pathExperimentA;
                break;
            } else if (headers.cookie[i].value.indexOf(cookieExperimentB) >= 0) {
                console.log('Experiment B cookie found');
                experimentUri = pathExperimentB;
                break;
            }
        }
    }

    if (!experimentUri) {
        console.log('Experiment cookie has not been found. Throwing dice...');
        if (Math.random() < 0.75) {
            experimentUri = pathExperimentA;
        } else {
            experimentUri = pathExperimentB;
        }
    }

    request.uri = experimentUri;
    console.log(`Request uri set to "${request.uri}"`);
    callback(null, request);
};
```

Python

```
import json
import random

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    headers = request['headers']
```

```
    if request['uri'] != '/experiment-pixel.jpg':
        # Not an A/B Test
        return request

    cookieExperimentA, cookieExperimentB = 'X-Experiment-Name=A', 'X-Experiment-
Name=B'
    pathExperimentA, pathExperimentB = '/experiment-group/control-pixel.jpg', '/
experiment-group/treatment-pixel.jpg'

    '''
    Lambda at the Edge headers are array objects.

    When a client sends multiple headers with the same name, each
    header is a separate entry in the array. For example:
    > GET /viewerRes/test HTTP/1.1
    > X-Custom: valueA
    > X-Custom: valueB

    You can access the first at headers["x-custom"][0].value
    and the second at headers["x-custom"][1].value.

    CloudFront concatenates multiple Cookie headers into a single entry, separated
 by "; ".
    For example:
    > Cookie: First=1; Second=2
    > Cookie: ClientCode=abc

    becomes headers["cookie"][0].value = "First=1; Second=2; ClientCode=abc"
    This behavior is consistent across HTTP/1.1, HTTP/2, and HTTP/3.

    Header values are not parsed.
    '''

    experimentUri = ""

    for cookie in headers.get('cookie', []):
        if cookieExperimentA in cookie['value']:
            print("Experiment A cookie found")
            experimentUri = pathExperimentA
            break
        elif cookieExperimentB in cookie['value']:
            print("Experiment B cookie found")
            experimentUri = pathExperimentB
```

```
            break

    if not experimentUri:
        print("Experiment cookie has not been found. Throwing dice...")
        if random.random() < 0.75:
            experimentUri = pathExperimentA
        else:
            experimentUri = pathExperimentB

    request['uri'] = experimentUri
    print(f"Request uri set to {experimentUri}")
    return request
```

Example: Override a response header

The following example shows how to change the value of a response header based on the value of another header.

Node.js

```
export const handler = async (event) => {
    const response = event.Records[0].cf.response;
    const headers = response.headers;

    const headerNameSrc = 'X-Amz-Meta-Last-Modified';
    const headerNameDst = 'Last-Modified';

    if (headers[headerNameSrc.toLowerCase()]) {
        headers[headerNameDst.toLowerCase()] = [{
            key: headerNameDst,
            value: headers[headerNameSrc.toLowerCase()][0].value,
        }];
        console.log(`Response header "${headerNameDst}" was set to ` +
                    `"${headers[headerNameDst.toLowerCase()][0].value}"`);
    }

    return response;
};
```

Python

```
import json
```

```
def lambda_handler(event, context):
    response = event['Records'][0]['cf']['response']
    headers = response['headers']

    header_name_src = 'X-Amz-Meta-Last-Modified'
    header_name_dst = 'Last-Modified'

    if headers.get(header_name_src.lower()):
        headers[header_name_dst.lower()] = [{
            'key': header_name_dst,
            'value': headers[header_name_src.lower()][0]['value']
        }]
        print(f'Response header "{header_name_dst}" was set to '
              f'"{headers[header_name_dst.lower()][0]["value"]}"')

    return response
```

Generate responses - examples

The following examples show how you can use Lambda@Edge to generate responses.

Topics

- Example: Serve static content (generated response)
- Example: Generate an HTTP redirect (generated response)

Example: Serve static content (generated response)

The following example shows how to use a Lambda function to serve static website content, which reduces the load on the origin server and reduces overall latency.

> **Note**
>
> You can generate HTTP responses for viewer request and origin request events. For more information, see the section called “Generate HTTP responses in request triggers”.
> You can also replace or remove the body of the HTTP response in origin response events. For more information, see the section called “Update HTTP responses in origin response triggers”.

Node.js

```
'use strict';

const content = `
<\!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Simple Lambda@Edge Static Content Response</title>
  </head>
  <body>
    <p>Hello from Lambda@Edge!</p>
  </body>
</html>
`;

exports.handler = (event, context, callback) => {
    /*
     * Generate HTTP OK response using 200 status code with HTML body.
     */
    const response = {
        status: '200',
        statusDescription: 'OK',
        headers: {
            'cache-control': [{
                key: 'Cache-Control',
                value: 'max-age=100'
            }],
            'content-type': [{
                key: 'Content-Type',
                value: 'text/html'
            }]
        },
        body: content,
    };
    callback(null, response);
};
```

Python

```
import json
```

```
CONTENT = """
<\!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Simple Lambda@Edge Static Content Response</title>
</head>
<body>
    <p>Hello from Lambda@Edge!</p>
</body>
</html>
"""

def lambda_handler(event, context):
    # Generate HTTP OK response using 200 status code with HTML body.
    response = {
        'status': '200',
        'statusDescription': 'OK',
        'headers': {
            'cache-control': [
                {
                    'key': 'Cache-Control',
                    'value': 'max-age=100'
                }
            ],
            "content-type": [
                {
                    'key': 'Content-Type',
                    'value': 'text/html'
                }
            ]
        },
        'body': CONTENT
    }
    return response
```

Example: Generate an HTTP redirect (generated response)

The following example shows how to generate an HTTP redirect.

Note

You can generate HTTP responses for viewer request and origin request events. For more information, see Generate HTTP responses in request triggers.

Node.js

```
'use strict';

exports.handler = (event, context, callback) => {
    /*
     * Generate HTTP redirect response with 302 status code and Location header.
     */
    const response = {
        status: '302',
        statusDescription: 'Found',
        headers: {
            location: [{
                key: 'Location',
                value: 'https://docs.aws.amazon.com/lambda/latest/dg/lambda-
edge.html',
            }],
        },
    };
    callback(null, response);
};
```

Python

```
def lambda_handler(event, context):

    # Generate HTTP redirect response with 302 status code and Location header.

    response = {
        'status': '302',
        'statusDescription': 'Found',
        'headers': {
            'location': [{
                'key': 'Location',
                'value': 'https://docs.aws.amazon.com/lambda/latest/dg/lambda-
edge.html'
```

```
            }]
        }
    }

    return response
```

Query strings - examples

The following examples show ways that you can use Lambda@Edge with query strings.

Topics

- Example: Add a header based on a query string parameter
- Example: Normalize query string parameters to improve the cache hit ratio
- Example: Redirect unauthenticated users to a sign-in page

Example: Add a header based on a query string parameter

The following example shows how to get the key-value pair of a query string parameter, and then add a header based on those values.

Node.js

```
'use strict';

const querystring = require('querystring');
exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;

    /* When a request contains a query string key-value pair but the origin server
     * expects the value in a header, you can use this Lambda function to
     * convert the key-value pair to a header. Here's what the function does:
     * 1. Parses the query string and gets the key-value pair.
     * 2. Adds a header to the request using the key-value pair that the function
 got in step 1.
     */

    /* Parse request querystring to get javascript object */
    const params = querystring.parse(request.querystring);

    /* Move auth param from querystring to headers */
```

```
    const headerName = 'Auth-Header';
    request.headers[headerName.toLowerCase()] = [{ key: headerName, value:
 params.auth }];
    delete params.auth;

    /* Update request querystring */
    request.querystring = querystring.stringify(params);

    callback(null, request);
};
```

Python

```
from urllib.parse import parse_qs, urlencode

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']

    '''
    When a request contains a query string key-value pair but the origin server
    expects the value in a header, you can use this Lambda function to
    convert the key-value pair to a header. Here's what the function does:
        1. Parses the query string and gets the key-value pair.
        2. Adds a header to the request using the key-value pair that the function
 got in step 1.
    '''

    # Parse request querystring to get dictionary/json
    params = {k : v[0] for k, v in parse_qs(request['querystring']).items()}

    # Move auth param from querystring to headers
    headerName = 'Auth-Header'
    request['headers'][headerName.lower()] = [{'key': headerName, 'value':
 params['auth']}]
    del params['auth']

    # Update request querystring
    request['querystring'] = urlencode(params)

    return request
```

Example: Normalize query string parameters to improve the cache hit ratio

The following example shows how to improve your cache hit ratio by making the following changes to query strings before CloudFront forwards requests to your origin:

- Alphabetize key-value pairs by the name of the parameter.
- Change the case of key-value pairs to lowercase.

For more information, see Cache content based on query string parameters.

Node.js

```
'use strict';

const querystring = require('querystring');

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    /* When you configure a distribution to forward query strings to the origin and
     * to cache based on an allowlist of query string parameters, we recommend
     * the following to improve the cache-hit ratio:
     * - Always list parameters in the same order.
     * - Use the same case for parameter names and values.
     *
     * This function normalizes query strings so that parameter names and values
     * are lowercase and parameter names are in alphabetical order.
     *
     * For more information, see:
     * https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/
QueryStringParameters.html
     */

    console.log('Query String: ', request.querystring);

    /* Parse request query string to get javascript object */
    const params = querystring.parse(request.querystring.toLowerCase());
    const sortedParams = {};

    /* Sort param keys */
    Object.keys(params).sort().forEach(key => {
        sortedParams[key] = params[key];
    });
```

```
    /* Update request querystring with normalized  */
    request.querystring = querystring.stringify(sortedParams);

    callback(null, request);
};
```

Python

```
from urllib.parse import parse_qs, urlencode

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    '''
    When you configure a distribution to forward query strings to the origin and
    to cache based on an allowlist of query string parameters, we recommend
    the following to improve the cache-hit ratio:
    Always list parameters in the same order.
    - Use the same case for parameter names and values.

    This function normalizes query strings so that parameter names and values
    are lowercase and parameter names are in alphabetical order.

    For more information, see:
    https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/
QueryStringParameters.html
    '''
    print("Query string: ", request["querystring"])

    # Parse request query string to get js object
    params = {k : v[0] for k, v in parse_qs(request['querystring'].lower()).items()}

    # Sort param keys
    sortedParams = sorted(params.items(), key=lambda x: x[0])

    # Update request querystring with normalized
    request['querystring'] = urlencode(sortedParams)

    return request
```

Example: Redirect unauthenticated users to a sign-in page

The following example shows how to redirect users to a sign-in page if they haven't entered their credentials.

Node.js

```
'use strict';

function parseCookies(headers) {
    const parsedCookie = {};
    if (headers.cookie) {
        headers.cookie[0].value.split(';').forEach((cookie) => {
            if (cookie) {
                const parts = cookie.split('=');
                parsedCookie[parts[0].trim()] = parts[1].trim();
            }
        });
    }
    return parsedCookie;
}

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    /* Check for session-id in request cookie in viewer-request event,
     * if session-id is absent, redirect the user to sign in page with original
     * request sent as redirect_url in query params.
     */

    /* Check for session-id in cookie, if present then proceed with request */
    const parsedCookies = parseCookies(headers);
    if (parsedCookies && parsedCookies['session-id']) {
        callback(null, request);
        return;
    }

    /* URI encode the original request to be sent as redirect_url in query params */
    const encodedRedirectUrl = encodeURIComponent(`https://
${headers.host[0].value}${request.uri}?${request.querystring}`);
    const response = {
        status: '302',
        statusDescription: 'Found',
```

```
        headers: {
            location: [{
                key: 'Location',
                value: `https://www.example.com/signin?redirect_url=
${encodedRedirectUrl}`,
            }],
        },
    };
    callback(null, response);
};
```

Python

```
import urllib

def parseCookies(headers):
    parsedCookie = {}
    if headers.get('cookie'):
        for cookie in headers['cookie'][0]['value'].split(';'):
            if cookie:
                parts = cookie.split('=')
                parsedCookie[parts[0].strip()] = parts[1].strip()
    return parsedCookie

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    headers = request['headers']

    '''
    Check for session-id in request cookie in viewer-request event,
    if session-id is absent, redirect the user to sign in page with original
    request sent as redirect_url in query params.
    '''

    # Check for session-id in cookie, if present, then proceed with request
    parsedCookies = parseCookies(headers)

    if parsedCookies and parsedCookies['session-id']:
        return request

    # URI encode the original request to be sent as redirect_url in query params
    redirectUrl = "https://%s%s?%s" % (headers['host'][0]['value'], request['uri'],
 request['querystring'])
```

```
    encodedRedirectUrl = urllib.parse.quote_plus(redirectUrl.encode('utf-8'))

    response = {
        'status': '302',
        'statusDescription': 'Found',
        'headers': {
            'location': [{
                'key': 'Location',
                'value': 'https://www.example.com/signin?redirect_url=%s' %
encodedRedirectUrl
            }]
        }
    }
    return response
```

Personalize content by country or device type headers - examples

The following examples show how you can use Lambda@Edge to customize behavior based on location or the type of device used by the viewer.

Topics

- Example: Redirect viewer requests to a country-specific URL
- Example: Serve different versions of an object based on the device

Example: Redirect viewer requests to a country-specific URL

The following example shows how to generate an HTTP redirect response with a country-specific URL and return the response to the viewer. This is useful when you want to provide country-specific responses. For example:

- If you have country-specific subdomains, such as us.example.com and tw.example.com, you can generate a redirect response when a viewer requests example.com.
- If you're streaming video but you don't have rights to stream the content in a specific country, you can redirect users in that country to a page that explains why they can't view the video.

Note the following:

- You must configure your distribution to cache based on the `CloudFront-Viewer-Country` header. For more information, see Cache based on selected request headers.
- CloudFront adds the `CloudFront-Viewer-Country` header after the viewer request event. To use this example, you must create a trigger for the origin request event.

Node.js

```
'use strict';

/* This is an origin request function */
exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    /*
     * Based on the value of the CloudFront-Viewer-Country header, generate an
     * HTTP status code 302 (Redirect) response, and return a country-specific
     * URL in the Location header.
     * NOTE: 1. You must configure your distribution to cache based on the
     *          CloudFront-Viewer-Country header. For more information, see
     *          https://docs.aws.amazon.com/console/cloudfront/cache-on-selected-
headers
     *       2. CloudFront adds the CloudFront-Viewer-Country header after the
 viewer
     *          request event. To use this example, you must create a trigger for
 the
     *          origin request event.
     */

    let url = 'https://example.com/';
    if (headers['cloudfront-viewer-country']) {
        const countryCode = headers['cloudfront-viewer-country'][0].value;
        if (countryCode === 'TW') {
            url = 'https://tw.example.com/';
        } else if (countryCode === 'US') {
            url = 'https://us.example.com/';
        }
    }

    const response = {
        status: '302',
        statusDescription: 'Found',
```

```
        headers: {
            location: [{
                key: 'Location',
                value: url,
            }],
        },
    };
    callback(null, response);
};
```

Python

```
# This is an origin request function

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    headers = request['headers']

    '''
    Based on the value of the CloudFront-Viewer-Country header, generate an
    HTTP status code 302 (Redirect) response, and return a country-specific
    URL in the Location header.
    NOTE: 1. You must configure your distribution to cache based on the
            CloudFront-Viewer-Country header. For more information, see
            https://docs.aws.amazon.com/console/cloudfront/cache-on-selected-headers
          2. CloudFront adds the CloudFront-Viewer-Country header after the viewer
            request event. To use this example, you must create a trigger for the
            origin request event.
    '''

    url = 'https://example.com/'
    viewerCountry = headers.get('cloudfront-viewer-country')
    if viewerCountry:
        countryCode = viewerCountry[0]['value']
        if countryCode == 'TW':
            url = 'https://tw.example.com/'
        elif countryCode == 'US':
            url = 'https://us.example.com/'

    response = {
        'status': '302',
        'statusDescription': 'Found',
        'headers': {
```

```
            'location': [{
                'key': 'Location',
                'value': url
            }]
        }
    }

    return response
```

Example: Serve different versions of an object based on the device

The following example shows how to serve different versions of an object based on the type of device that the user is using, for example, a mobile device or a tablet. Note the following:

- You must configure your distribution to cache based on the `CloudFront-Is-*-Viewer` headers. For more information, see Cache based on selected request headers.
- CloudFront adds the `CloudFront-Is-*-Viewer` headers after the viewer request event. To use this example, you must create a trigger for the origin request event.

Node.js

```
'use strict';

/* This is an origin request function */
exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    /*
     * Serve different versions of an object based on the device type.
     * NOTE: 1. You must configure your distribution to cache based on the
     *          CloudFront-Is-*-Viewer headers. For more information, see
     *          the following documentation:
     *          https://docs.aws.amazon.com/console/cloudfront/cache-on-selected-
headers
     *          https://docs.aws.amazon.com/console/cloudfront/cache-on-device-type
     *       2. CloudFront adds the CloudFront-Is-*-Viewer headers after the viewer
     *          request event. To use this example, you must create a trigger for
 the
     *          origin request event.
     */
```

```
    const desktopPath = '/desktop';
    const mobilePath = '/mobile';
    const tabletPath = '/tablet';
    const smarttvPath = '/smarttv';

    if (headers['cloudfront-is-desktop-viewer']
        && headers['cloudfront-is-desktop-viewer'][0].value === 'true') {
        request.uri = desktopPath + request.uri;
    } else if (headers['cloudfront-is-mobile-viewer']
               && headers['cloudfront-is-mobile-viewer'][0].value === 'true') {
        request.uri = mobilePath + request.uri;
    } else if (headers['cloudfront-is-tablet-viewer']
               && headers['cloudfront-is-tablet-viewer'][0].value === 'true') {
        request.uri = tabletPath + request.uri;
    } else if (headers['cloudfront-is-smarttv-viewer']
               && headers['cloudfront-is-smarttv-viewer'][0].value === 'true') {
        request.uri = smarttvPath + request.uri;
    }
    console.log(`Request uri set to "${request.uri}"`);

    callback(null, request);
};
```

Python

```
# This is an origin request function
def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    headers = request['headers']

    '''
    Serve different versions of an object based on the device type.
    NOTE: 1. You must configure your distribution to cache based on the
            CloudFront-Is-*-Viewer headers. For more information, see
            the following documentation:
            https://docs.aws.amazon.com/console/cloudfront/cache-on-selected-headers
            https://docs.aws.amazon.com/console/cloudfront/cache-on-device-type
          2. CloudFront adds the CloudFront-Is-*-Viewer headers after the viewer
            request event. To use this example, you must create a trigger for the
            origin request event.
    '''
```

```
    desktopPath = '/desktop';
    mobilePath = '/mobile';
    tabletPath = '/tablet';
    smarttvPath = '/smarttv';

    if 'cloudfront-is-desktop-viewer' in headers and headers['cloudfront-is-desktop-
viewer'][0]['value'] == 'true':
        request['uri'] = desktopPath + request['uri']
    elif 'cloudfront-is-mobile-viewer' in headers and headers['cloudfront-is-mobile-
viewer'][0]['value'] == 'true':
        request['uri'] = mobilePath + request['uri']
    elif 'cloudfront-is-tablet-viewer' in headers and headers['cloudfront-is-tablet-
viewer'][0]['value'] == 'true':
        request['uri'] = tabletPath + request['uri']
    elif 'cloudfront-is-smarttv-viewer' in headers and headers['cloudfront-is-
smarttv-viewer'][0]['value'] == 'true':
        request['uri'] = smarttvPath + request['uri']

    print("Request uri set to %s" % request['uri'])

    return request
```

Content-based dynamic origin selection - examples

The following examples show how you can use Lambda@Edge to route to different origins based on information in the request.

Topics

- Example: Use an origin request trigger to change from a custom origin to an Amazon S3 origin
- Example: Use an origin-request trigger to change the Amazon S3 origin Region
- Example: Use an origin request trigger to change from an Amazon S3 origin to a custom origin
- Example: Use an origin request trigger to gradually transfer traffic from one Amazon S3 bucket to another
- Example: Use an origin request trigger to change the origin domain name based on the country header

Example: Use an origin request trigger to change from a custom origin to an Amazon S3 origin

This function demonstrates how an origin-request trigger can be used to change from a custom origin to an Amazon S3 origin from which the content is fetched, based on request properties.

Node.js

```
'use strict';

 const querystring = require('querystring');

 exports.handler = (event, context, callback) => {
     const request = event.Records[0].cf.request;

     /**
      * Reads query string to check if S3 origin should be used, and
      * if true, sets S3 origin properties.
      */

     const params = querystring.parse(request.querystring);

     if (params['useS3Origin']) {
         if (params['useS3Origin'] === 'true') {
             const s3DomainName = 'amzn-s3-demo-bucket.s3.amazonaws.com';

             /* Set S3 origin fields */
             request.origin = {
                 s3: {
                     domainName: s3DomainName,
                     region: '',
                     authMethod: 'origin-access-identity',
                     path: '',
                     customHeaders: {}
                 }
             };
             request.headers['host'] = [{ key: 'host', value: s3DomainName}];
         }
     }

    callback(null, request);
};
```

Python

```
from urllib.parse import parse_qs

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    '''
    Reads query string to check if S3 origin should be used, and
    if true, sets S3 origin properties
    '''
    params = {k: v[0] for k, v in parse_qs(request['querystring']).items()}
    if params.get('useS3Origin') == 'true':
        s3DomainName = 'amzn-s3-demo-bucket.s3.amazonaws.com'

        # Set S3 origin fields
        request['origin'] = {
            's3': {
                'domainName': s3DomainName,
                'region': '',
                'authMethod': 'origin-access-identity',
                'path': '',
                'customHeaders': {}
            }
        }
        request['headers']['host'] = [{'key': 'host', 'value': s3DomainName}]
    return request
```

Example: Use an origin-request trigger to change the Amazon S3 origin Region

This function demonstrates how an origin-request trigger can be used to change the Amazon S3 origin from which the content is fetched, based on request properties.

In this example, you use the value of the `CloudFront-Viewer-Country` header to update the S3 bucket domain name to a bucket in a Region that is closer to the viewer. This can be useful in several ways:

- It reduces latencies when the Region specified is nearer to the viewer's country.
- It provides data sovereignty by making sure that data is served from an origin that's in the same country that the request came from.

To use this example, you must do the following:

- Configure your distribution to cache based on the `CloudFront-Viewer-Country` header. For more information, see Cache based on selected request headers.
- Create a trigger for this function in the origin request event. CloudFront adds the `CloudFront-Viewer-Country` header after the viewer request event, so to use this example, you must make sure that the function executes for an origin request.

> **Note**
>
> The following example code uses the same origin access identity (OAI) for all S3 buckets that you're using for your origin. For more information, see Origin access identity.

Node.js

```
'use strict';

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;

    /**
     * This blueprint demonstrates how an origin-request trigger can be used to
     * change the origin from which the content is fetched, based on request
 properties.
     * In this example, we use the value of the CloudFront-Viewer-Country header
     * to update the S3 bucket domain name to a bucket in a Region that is closer to
     * the viewer.
     *
     * This can be useful in several ways:
     *      1) Reduces latencies when the Region specified is nearer to the viewer's
     *         country.
     *      2) Provides data sovereignty by making sure that data is served from an
     *         origin that's in the same country that the request came from.
     *
     * NOTE: 1. You must configure your distribution to cache based on the
     *          CloudFront-Viewer-Country header. For more information, see
     *          https://docs.aws.amazon.com/console/cloudfront/cache-on-selected-
headers
     *       2. CloudFront adds the CloudFront-Viewer-Country header after the
 viewer
```

```
     *          request event. To use this example, you must create a trigger for
 the
     *          origin request event.
     */

    const countryToRegion = {
        'DE': 'eu-central-1',
        'IE': 'eu-west-1',
        'GB': 'eu-west-2',
        'FR': 'eu-west-3',
        'JP': 'ap-northeast-1',
        'IN': 'ap-south-1'
    };

    if (request.headers['cloudfront-viewer-country']) {
        const countryCode = request.headers['cloudfront-viewer-country'][0].value;
        const region = countryToRegion[countryCode];

        /**
         * If the viewer's country is not in the list you specify, the request
         * goes to the default S3 bucket you've configured.
         */
        if (region) {
            /**
             * If you've set up OAI, the bucket policy in the destination bucket
             * should allow the OAI GetObject operation, as configured by default
             * for an S3 origin with OAI. Another requirement with OAI is to provide
             * the Region so it can be used for the SIGV4 signature. Otherwise, the
             * Region is not required.
             */
            request.origin.s3.region = region;
            const domainName = `amzn-s3-demo-bucket-in-${region}.s3.
${region}.amazonaws.com`;
            request.origin.s3.domainName = domainName;
            request.headers['host'] = [{ key: 'host', value: domainName }];
        }
    }

    callback(null, request);
};
```

Python

```
def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']

    '''
    This blueprint demonstrates how an origin-request trigger can be used to
    change the origin from which the content is fetched, based on request
 properties.
    In this example, we use the value of the CloudFront-Viewer-Country header
    to update the S3 bucket domain name to a bucket in a Region that is closer to
    the viewer.

    This can be useful in several ways:
        1) Reduces latencies when the Region specified is nearer to the viewer's
            country.
        2) Provides data sovereignty by making sure that data is served from an
            origin that's in the same country that the request came from.

    NOTE: 1. You must configure your distribution to cache based on the
            CloudFront-Viewer-Country header. For more information, see
            https://docs.aws.amazon.com/console/cloudfront/cache-on-selected-headers
          2. CloudFront adds the CloudFront-Viewer-Country header after the viewer
            request event. To use this example, you must create a trigger for the
            origin request event.
    '''

    countryToRegion = {
        'DE': 'eu-central-1',
        'IE': 'eu-west-1',
        'GB': 'eu-west-2',
        'FR': 'eu-west-3',
        'JP': 'ap-northeast-1',
        'IN': 'ap-south-1'
    }

    viewerCountry = request['headers'].get('cloudfront-viewer-country')
    if viewerCountry:
        countryCode = viewerCountry[0]['value']
        region = countryToRegion.get(countryCode)

        # If the viewer's country in not in the list you specify, the request
        # goes to the default S3 bucket you've configured
        if region:
```

```
            '''
            If you've set up OAI, the bucket policy in the destination bucket
            should allow the OAI GetObject operation, as configured by default
            for an S3 origin with OAI. Another requirement with OAI is to provide
            the Region so it can be used for the SIGV4 signature. Otherwise, the
            Region is not required.
            '''
            request['origin']['s3']['region'] = region
            domainName = 'amzn-s3-demo-bucket-in-{0}.s3.
{0}.amazonaws.com'.format(region)
            request['origin']['s3']['domainName'] = domainName
            request['headers']['host'] = [{'key': 'host', 'value': domainName}]

    return request
```

Example: Use an origin request trigger to change from an Amazon S3 origin to a custom origin

This function demonstrates how an origin-request trigger can be used to change the custom origin from which the content is fetched, based on request properties.

Node.js

```
'use strict';

const querystring = require('querystring');

 exports.handler = (event, context, callback) => {
     const request = event.Records[0].cf.request;

     /**
      * Reads query string to check if custom origin should be used, and
      * if true, sets custom origin properties.
      */

     const params = querystring.parse(request.querystring);

     if (params['useCustomOrigin']) {
         if (params['useCustomOrigin'] === 'true') {

             /* Set custom origin fields*/
             request.origin = {
                 custom: {
```

```
                    domainName: 'www.example.com',
                    port: 443,
                    protocol: 'https',
                    path: '',
                    sslProtocols: ['TLSv1', 'TLSv1.1'],
                    readTimeout: 5,
                    keepaliveTimeout: 5,
                    customHeaders: {}
                }
            };
            request.headers['host'] = [{ key: 'host', value: 'www.example.com'}];
        }
    }
    callback(null, request);
};
```

Python

```
from urllib.parse import parse_qs

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']

    # Reads query string to check if custom origin should be used, and
    # if true, sets custom origin properties

    params = {k: v[0] for k, v in parse_qs(request['querystring']).items()}

    if params.get('useCustomOrigin') == 'true':
            # Set custom origin fields
            request['origin'] = {
                'custom': {
                    'domainName': 'www.example.com',
                    'port': 443,
                    'protocol': 'https',
                    'path': '',
                    'sslProtocols': ['TLSv1', 'TLSv1.1'],
                    'readTimeout': 5,
                    'keepaliveTimeout': 5,
                    'customHeaders': {}
                }
            }
```

```
            request['headers']['host'] = [{'key': 'host', 'value':
 'www.example.com'}]

    return request
```

Example: Use an origin request trigger to gradually transfer traffic from one Amazon S3 bucket to another

This function demonstrates how you can gradually transfer traffic from one Amazon S3 bucket to another in a controlled way.

Node.js

```
'use strict';

    function getRandomInt(min, max) {
        /* Random number is inclusive of min and max*/
        return Math.floor(Math.random() * (max - min + 1)) + min;
 }

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const BLUE_TRAFFIC_PERCENTAGE = 80;

    /**
      * This Lambda function demonstrates how to gradually transfer traffic from
      * one S3 bucket to another in a controlled way.
      * We define a variable BLUE_TRAFFIC_PERCENTAGE which can take values from
      * 1 to 100. If the generated randomNumber less than or equal to
 BLUE_TRAFFIC_PERCENTAGE, traffic
      * is re-directed to blue-bucket. If not, the default bucket that we've
 configured
      * is used.
      */

    const randomNumber = getRandomInt(1, 100);

if (randomNumber <= BLUE_TRAFFIC_PERCENTAGE) {
          const domainName = 'blue-bucket.s3.amazonaws.com';
          request.origin.s3.domainName = domainName;
          request.headers['host'] = [{ key: 'host', value: domainName}];
     }
```

```
    callback(null, request);
};
```

Python

```
import math
import random

def getRandomInt(min, max):
    # Random number is inclusive of min and max
    return math.floor(random.random() * (max - min + 1)) + min

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    BLUE_TRAFFIC_PERCENTAGE = 80

    '''
    This Lambda function demonstrates how to gradually transfer traffic from
    one S3 bucket to another in a controlled way.
    We define a variable BLUE_TRAFFIC_PERCENTAGE which can take values from
    1 to 100. If the generated randomNumber less than or equal to
 BLUE_TRAFFIC_PERCENTAGE, traffic
    is re-directed to blue-bucket. If not, the default bucket that we've configured
    is used.
    '''

    randomNumber = getRandomInt(1, 100)

    if randomNumber <= BLUE_TRAFFIC_PERCENTAGE:
        domainName = 'blue-bucket.s3.amazonaws.com'
        request['origin']['s3']['domainName'] = domainName
        request['headers']['host'] = [{'key': 'host', 'value': domainName}]

    return request
```

Example: Use an origin request trigger to change the origin domain name based on the country header

This function demonstrates how you can change the origin domain name based on the `CloudFront-Viewer-Country` header, so content is served from an origin closer to the viewer's country.

Implementing this functionality for your distribution can have advantages such as the following:

- Reducing latencies when the Region specified is nearer to the viewer's country
- Providing data sovereignty by making sure that data is served from an origin that's in the same country that the request came from

Note that to enable this functionality you must configure your distribution to cache based on the `CloudFront-Viewer-Country` header. For more information, see the section called “Cache based on selected request headers”.

Node.js

```
'use strict';

exports.handler = (event, context, callback) => {
     const request = event.Records[0].cf.request;

  if (request.headers['cloudfront-viewer-country']) {
        const countryCode = request.headers['cloudfront-viewer-country'][0].value;
        if (countryCode === 'GB' || countryCode === 'DE' || countryCode === 'IE' )
 {
            const domainName = 'eu.example.com';
            request.origin.custom.domainName = domainName;
            request.headers['host'] = [{key: 'host', value: domainName}];
        }
     }

    callback(null, request);
};
```

Python

```
def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']

    viewerCountry = request['headers'].get('cloudfront-viewer-country')
    if viewerCountry:
        countryCode = viewerCountry[0]['value']
        if countryCode == 'GB' or countryCode == 'DE' or countryCode == 'IE':
            domainName = 'eu.example.com'
            request['origin']['custom']['domainName'] = domainName
```

```
            request['headers']['host'] = [{'key': 'host', 'value': domainName}]
    return request
```

Update error statuses - examples

The following examples provide guidance for how you can use Lambda@Edge to change the error status that is returned to users.

Topics

- Example: Use an origin response trigger to update the error status code to 200
- Example: Use an origin response trigger to update the error status code to 302

Example: Use an origin response trigger to update the error status code to 200

This function demonstrates how you can update the response status to 200 and generate static body content to return to the viewer in the following scenario:

- The function is triggered in an origin response.
- The response status from the origin server is an error status code (4xx or 5xx).

Node.js

```
'use strict';

exports.handler = (event, context, callback) => {
    const response = event.Records[0].cf.response;

    /**
     * This function updates the response status to 200 and generates static
     * body content to return to the viewer in the following scenario:
     * 1. The function is triggered in an origin response
     * 2. The response status from the origin server is an error status code (4xx or
 5xx)
     */

    if (response.status >= 400 && response.status <= 599) {
        response.status = 200;
        response.statusDescription = 'OK';
        response.body = 'Body generation example';
```

```
    }

    callback(null, response);
};
```

Python

```
def lambda_handler(event, context):
    response = event['Records'][0]['cf']['response']

    '''
    This function updates the response status to 200 and generates static
    body content to return to the viewer in the following scenario:
    1. The function is triggered in an origin response
    2. The response status from the origin server is an error status code (4xx or
 5xx)
    '''

    if int(response['status']) >= 400 and int(response['status']) <= 599:
        response['status'] = 200
        response['statusDescription'] = 'OK'
        response['body'] = 'Body generation example'
    return response
```

Example: Use an origin response trigger to update the error status code to 302

This function demonstrates how you can update the HTTP status code to 302 to redirect to another path (cache behavior) that has a different origin configured. Note the following:

- The function is triggered in an origin response.
- The response status from the origin server is an error status code (4xx or 5xx).

Node.js

```
'use strict';

exports.handler = (event, context, callback) => {
    const response = event.Records[0].cf.response;
    const request = event.Records[0].cf.request;

    /**
```

```
    * This function updates the HTTP status code in the response to 302, to
 redirect to another
    * path (cache behavior) that has a different origin configured. Note the
 following:
    * 1. The function is triggered in an origin response
    * 2. The response status from the origin server is an error status code (4xx or
 5xx)
    */

   if (response.status >= 400 && response.status <= 599) {
       const redirect_path = `/plan-b/path?${request.querystring}`;

       response.status = 302;
       response.statusDescription = 'Found';

       /* Drop the body, as it is not required for redirects */
       response.body = '';
       response.headers['location'] = [{ key: 'Location', value: redirect_path }];
   }

   callback(null, response);
};
```

Python

```
def lambda_handler(event, context):
    response = event['Records'][0]['cf']['response']
    request = event['Records'][0]['cf']['request']

    '''
    This function updates the HTTP status code in the response to 302, to redirect
 to another
    path (cache behavior) that has a different origin configured. Note the
 following:
    1. The function is triggered in an origin response
    2. The response status from the origin server is an error status code (4xx or
 5xx)
    '''

    if int(response['status']) >= 400 and int(response['status']) <= 599:
        redirect_path = '/plan-b/path?%s' % request['querystring']

        response['status'] = 302
```

```
        response['statusDescription'] = 'Found'

        # Drop the body as it is not required for redirects
        response['body'] = ''
        response['headers']['location'] = [{'key': 'Location', 'value':
redirect_path}]

    return response
```

Access the request body - examples

The following examples show how you can use Lambda@Edge to work with POST requests.

> **Note**
>
> To use these examples, you must enable the *include body* option in the distribution's Lambda function association. It is not enabled by default.
>
> - To enable this setting in the CloudFront console, select the check box for **Include Body** in the **Lambda Function Association**.
> - To enable this setting in the CloudFront API or with CloudFormation, set the `IncludeBody` field to `true` in `LambdaFunctionAssociation`.

Topics

- Example: Use a request trigger to read an HTML form
- Example: Use a request trigger to modify an HTML form

Example: Use a request trigger to read an HTML form

This function demonstrates how you can process the body of a POST request generated by an HTML form (web form), such as a "contact us" form. For example, you might have an HTML form like the following:

```
<html>
  <form action="https://example.com" method="post">
    Param 1: <input type="text" name="name1"><br>
    Param 2: <input type="text" name="name2"><br>
    <input type="submit" value="Submit">
```

```
  </form>
</html>
```

For the example function that follows, the function must be triggered in a CloudFront viewer request or origin request.

Node.js

```
'use strict';

const querystring = require('querystring');

/**
 * This function demonstrates how you can read the body of a POST request
 * generated by an HTML form (web form). The function is triggered in a
 * CloudFront viewer request or origin request event type.
 */

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;

    if (request.method === 'POST') {
        /* HTTP body is always passed as base64-encoded string. Decode it. */
        const body = Buffer.from(request.body.data, 'base64').toString();

        /* HTML forms send the data in query string format. Parse it. */
        const params = querystring.parse(body);

        /* For demonstration purposes, we only log the form fields here.
         * You can put your custom logic here. For example, you can store the
         * fields in a database, such as Amazon DynamoDB, and generate a response
         * right from your Lambda@Edge function.
         */
        for (let param in params) {
            console.log(`For "${param}" user submitted "${params[param]}".\n`);
        }
    }
    return callback(null, request);
};
```

Python

```
import base64
```

```
from urllib.parse import parse_qs

'''
Say there is a POST request body generated by an HTML such as:

<html>
<form action="https://example.com" method="post">
    Param 1: <input type="text" name="name1"><br>
    Param 2: <input type="text" name="name2"><br>
    <input type="submit" value="Submit">
</form>
</html>

'''

'''
This function demonstrates how you can read the body of a POST request
generated by an HTML form (web form). The function is triggered in a
CloudFront viewer request or origin request event type.
'''

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']

    if request['method'] == 'POST':
        # HTTP body is always passed as base64-encoded string. Decode it
        body = base64.b64decode(request['body']['data'])

        # HTML forms send the data in query string format. Parse it
        params = {k: v[0] for k, v in parse_qs(body).items()}

        '''
        For demonstration purposes, we only log the form fields here.
        You can put your custom logic here. For example, you can store the
        fields in a database, such as Amazon DynamoDB, and generate a response
        right from your Lambda@Edge function.
        '''
        for key, value in params.items():
            print("For %s use submitted %s" % (key, value))

    return request
```

Example: Use a request trigger to modify an HTML form

This function demonstrates how you can modify the body of a POST request generated by an HTML form (web form). The function is triggered in a CloudFront viewer request or origin request.

Node.js

```
'use strict';

const querystring = require('querystring');

exports.handler = (event, context, callback) => {
    var request = event.Records[0].cf.request;
    if (request.method === 'POST') {
        /* Request body is being replaced. To do this, update the following
        /* three fields:
         *    1) body.action to 'replace'
         *    2) body.encoding to the encoding of the new data.
         *
         *       Set to one of the following values:
         *
         *           text - denotes that the generated body is in text format.
         *               Lambda@Edge will propagate this as is.
         *           base64 - denotes that the generated body is base64 encoded.
         *               Lambda@Edge will base64 decode the data before sending
         *               it to the origin.
         *    3) body.data to the new body.
         */
        request.body.action = 'replace';
        request.body.encoding = 'text';
        request.body.data = getUpdatedBody(request);
    }
    callback(null, request);
};

function getUpdatedBody(request) {
    /* HTTP body is always passed as base64-encoded string. Decode it. */
    const body = Buffer.from(request.body.data, 'base64').toString();

    /* HTML forms send data in query string format. Parse it. */
    const params = querystring.parse(body);

    /* For demonstration purposes, we're adding one more param.
     *
```

```
     * You can put your custom logic here. For example, you can truncate long
     * bodies from malicious requests.
     */
    params['new-param-name'] = 'new-param-value';
    return querystring.stringify(params);
}
```

Python

```
import base64
from urllib.parse import parse_qs, urlencode

def lambda_handler(event, context):
    request = event['Records'][0]['cf']['request']
    if request['method'] == 'POST':
        '''
        Request body is being replaced. To do this, update the following
        three fields:
            1) body.action to 'replace'
            2) body.encoding to the encoding of the new data.

            Set to one of the following values:

                text - denotes that the generated body is in text format.
                    Lambda@Edge will propagate this as is.
                base64 - denotes that the generated body is base64 encoded.
                    Lambda@Edge will base64 decode the data before sending
                    it to the origin.
            3) body.data to the new body.
        '''
        request['body']['action'] = 'replace'
        request['body']['encoding'] = 'text'
        request['body']['data'] = getUpdatedBody(request)
    return request

def getUpdatedBody(request):
    # HTTP body is always passed as base64-encoded string. Decode it
    body = base64.b64decode(request['body']['data'])

    # HTML forms send data in query string format. Parse it
    params = {k: v[0] for k, v in parse_qs(body).items()}

    # For demonstration purposes, we're adding one more param
```

```
    # You can put your custom logic here. For example, you can truncate long
    # bodies from malicious requests
    params['new-param-name'] = 'new-param-value'
    return urlencode(params)
```

Restrictions on edge functions

The following topics describe the restrictions that apply to CloudFront Functions and Lambda@Edge. Some restrictions apply to all edge functions, while others apply only to CloudFront Functions or Lambda@Edge.

Each topic provides detailed information about the limitations and constraints you should consider when you develop and deploy edge functions with CloudFront.

Understanding these restrictions helps you make sure that your edge functions operate as expected and comply with the supported features.

Topics

- Restrictions on all edge functions
- Restrictions on CloudFront Functions
- Restrictions on Lambda@Edge

For information about quotas (formerly referred to as limits), see Quotas on CloudFront Functions and Quotas on Lambda@Edge.

Restrictions on all edge functions

The following restrictions apply to all edge functions, both CloudFront Functions and Lambda@Edge.

Topics

- AWS account ownership
- Combining CloudFront Functions with Lambda@Edge
- HTTP status codes
- HTTP headers

- Query strings
- URI
- URI, query string, and headers encoding
- Microsoft Smooth Streaming
- Tagging

AWS account ownership

To associate an edge function with a CloudFront distribution, the function and distribution must be owned by the same AWS account.

Combining CloudFront Functions with Lambda@Edge

For a given cache behavior, the following restrictions apply:

- Each event type (viewer request, origin request, origin response, and viewer response) can have only one edge function association.
- You can't combine CloudFront Functions and Lambda@Edge in viewer events (viewer request and viewer response).

All other combinations of edge functions are allowed. The following table explains the allowed combinations.

		CloudFront Functions	
		Viewer request	**Viewer response**
Lambda@Edge	**Viewer request**	Not allowed	Not allowed
	Origin request	Allowed	Allowed
	Origin response	Allowed	Allowed
	Viewer response	Not allowed	Not allowed

HTTP status codes

CloudFront doesn't invoke edge functions for viewer response events when the origin returns HTTP status code 400 or higher.

Lambda@Edge functions for origin response events are invoked for *all* origin responses, including when the origin returns HTTP status code 400 or higher. For more information, see Update HTTP responses in origin response triggers.

HTTP headers

Certain HTTP headers are disallowed, which means they're not exposed to edge functions and functions can't add them. Other headers are read-only, which means functions can read them but can't add, modify, or delete them.

Topics

- Disallowed headers
- Read-only headers

Disallowed headers

The following HTTP headers are not exposed to edge functions, and functions can't add them. If your function adds one of these headers, it fails CloudFront validation and CloudFront returns HTTP status code 502 (Bad Gateway) to the viewer.

- `Connection`
- `Expect`
- `Keep-Alive`
- `Proxy-Authenticate`
- `Proxy-Authorization`
- `Proxy-Connection`
- `Trailer`
- `Upgrade`
- `X-Accel-Buffering`
- `X-Accel-Charset`
- `X-Accel-Limit-Rate`

- X-Accel-Redirect
- X-Amz-Cf-*
- X-Amzn-Auth
- X-Amzn-Cf-Billing
- X-Amzn-Cf-Id
- X-Amzn-Cf-Xff
- X-Amzn-Errortype
- X-Amzn-Fle-Profile
- X-Amzn-Header-Count
- X-Amzn-Header-Order
- X-Amzn-Lambda-Integration-Tag
- X-Amzn-RequestId
- X-Cache
- X-Edge-*
- X-Forwarded-Proto
- X-Real-IP
- CloudFront-Viewer-Cert-PEM
- Client-Cert
- Client-Cert-Chain

Read-only headers

The following headers are read-only. Your function can read them and use them as input to the function logic, but it can't change the values. If your function adds or edits a read-only header, the request fails CloudFront validation and CloudFront returns HTTP status code 502 (Bad Gateway) to the viewer.

Read-only headers in viewer request events

The following headers are read-only in viewer request events.

- CDN-Loop
- Content-Length

- `Host`
- `Transfer-Encoding`
- `Via`

Read-only headers in origin request events (Lambda@Edge only)

The following headers are read-only in origin request events, which exist only in Lambda@Edge.

- `Accept-Encoding`
- `CDN-Loop`
- `Content-Length`
- `If-Modified-Since`
- `If-None-Match`
- `If-Range`
- `If-Unmodified-Since`
- `Transfer-Encoding`
- `Via`

Read-only headers in origin response events (Lambda@Edge only)

The following headers are read-only in origin response events, which exist only in Lambda@Edge.

- `Transfer-Encoding`
- `Via`

Read-only headers in viewer response events

The following headers are read-only in viewer response events for both CloudFront Functions and Lambda@Edge.

- `Warning`
- `Via`

The following headers are read-only in viewer response events for Lambda@Edge.

- `Content-Length`
- `Content-Encoding`
- `Transfer-Encoding`

Query strings

The following restrictions apply to functions that read, update, or create a query string in a request URI.

- (Lambda@Edge only) To access the query string in an origin request or origin response function, your cache policy or origin request policy must be set to **All** for **Query strings**.
- A function can create or update a query string for viewer request and origin request events (origin request events exist only in Lambda@Edge).
- A function can read a query string, but cannot create or update one, for origin response and viewer response events (origin response events exist only in Lambda@Edge).
- If a function creates or updates a query string, the following restrictions apply:
  - The query string can't include spaces, control characters, or the fragment identifier (#).
  - The total size of the URI, including the query string, must be less than 8,192 characters.
  - We recommend that you use percent encoding for the URI and query string. For more information, see URI, query string, and headers encoding.

URI

If a function changes the URI for a request, that doesn't change the cache behavior for the request or the origin that the request is forwarded to.

The total size of the URI, including the query string, must be less than 8,192 characters.

URI, query string, and headers encoding

The values for the URI, query string, and headers that are passed to edge functions are UTF-8 encoded. Your function should use UTF-8 encoding for the URI, query string, and header values that it returns. Percent encoding is compatible with UTF-8 encoding.

The following list explains how CloudFront handles encoding for the URI, query string, and headers:

- When values in the request are UTF-8 encoded, CloudFront forwards the values to your function without changing them.
- When values in the request are ISO-8859-1 encoded, CloudFront converts the values to UTF-8 encoding before forwarding them to your function.
- When values in the request are encoded using some other character encoding, CloudFront assumes that they're ISO-8859-1 encoded and tries to convert from ISO-8859-1 to UTF-8.

> **Important**
>
> The converted characters might be an inaccurate interpretation of the values in the original request. This might cause your function or your origin to produce an unintended result.

The values for the URI, query string, and headers that CloudFront forwards to your origin depend on whether a function changes the values:

- If a function doesn't change the URI, query string, or header, CloudFront forwards the values that it received in the request to your origin.
- If a function changes the URI, query string, or header, CloudFront forwards the UTF-8 encoded values.

Microsoft Smooth Streaming

You can't use edge functions with a CloudFront distribution that you're using for streaming media files that you've transcoded into the Microsoft Smooth Streaming format.

Tagging

You can't add tags to edge functions. For more information about tagging in CloudFront, see Tag a CloudFront resource.

Restrictions on CloudFront Functions

The following restrictions apply only to CloudFront Functions.

Contents

- Logs
- Request body
- Network calls
- Using temporary credentials with the CloudFront KeyValueStore API
- Runtime
- Compute utilization

For information about quotas (formerly referred to as limits), see Quotas on CloudFront Functions.

Logs

Function logs in CloudFront Functions are truncated at 10 KB.

Request body

CloudFront Functions can't access the body of the viewer's HTTP request. This is separate from the request body that your function can construct and send on a network call. For more information about network calls, see Helper methods for network calls.

Network calls

Your function can make one network call per invocation. The destination must be an origin that is configured on the same distribution as your function. For more information about network calls, see Helper methods for network calls.

Using temporary credentials with the CloudFront KeyValueStore API

You can use AWS Security Token Service (AWS STS) to generate temporary security credentials (also known as *session tokens*). Session tokens allow you to temporarily assume an AWS Identity and Access Management (IAM) role so that you can access AWS services.

To call the CloudFront KeyValueStore API, use a *Regional* endpoint in AWS STS to return a *version 2* session token. If you use the *global* endpoint for AWS STS (`sts.amazonaws.com`), AWS STS will generate a *version 1* session token, which isn't supported by Signature Version 4A (SigV4A). As a result, you will receive an authentication error.

To call the CloudFront KeyValueStore API, you can use the following options:

AWS CLI and AWS SDKs

You can configure the AWS CLI or an AWS SDK to use Regional AWS STS endpoints. For more information, see AWS STS Regionalized endpoints in the *AWS SDK and Tools Reference Guide*.

For more information about available AWS STS endpoints, see Regions and endpoints in the *IAM User Guide*.

SAML

You can configure SAML to use Regional AWS STS endpoints. For more information, see the How to use regional SAML endpoints for failover blog post.

`SetSecurityTokenServicePreferences` API

Instead of using a Regional AWS STS endpoint, you can configure the global endpoint for AWS STS to return version 2 session tokens. To do so, use the SetSecurityTokenServicePreferences API operation to configure your AWS account.

Example: IAM CLI command

```
aws iam set-security-token-service-preferences --global-endpoint-token-version
 v2Token
```

Tip

We recommend that you use the AWS STS Regional endpoints instead of this option. Regional endpoints provide higher availability and failover scenarios.

Custom identity provider

If you're using a custom identity provider that does the federation and assumes the role, use one of the previous options for the parent identity provider system that is responsible for generating the session token.

Runtime

The CloudFront Functions runtime environment doesn't support dynamic code evaluation, and it restricts access to the file system, environment variables, and timers. For more information, see Restricted features.

> **Note**
>
> To use CloudFront KeyValueStore, your CloudFront function must use JavaScript runtime 2.0.

Compute utilization

CloudFront Functions have a limit on the time they can take to run, measured as *compute utilization*. Compute utilization is a number between 0 and 100 that indicates the amount of time that the function took to run as a percentage of the maximum allowed time. For example, a compute utilization of 35 means that the function completed in 35% of the maximum allowed time.

When you test a function, you can see the compute utilization value in the output of the test event. For production functions, you can view the compute utilization metric on the Monitoring page in the CloudFront console, or in CloudWatch.

Restrictions on Lambda@Edge

The following restrictions apply only to Lambda@Edge.

Contents

- DNS resolution
- HTTP status codes
- Lambda function version
- Lambda Region
- Lambda role permissions
- Lambda features
- Supported runtimes
- CloudFront headers
- Restrictions on the request body with the include body option
- Response timeout and keep-alive timeout (custom origins only)

For information about quotas, see Quotas on Lambda@Edge.

DNS resolution

CloudFront performs a DNS resolution on the origin domain name *before* it executes your origin request Lambda@Edge function. If the DNS service for your domain is experiencing issues and CloudFront can't resolve the domain name to get the IP address, your Lambda@Edge function will not invoke. CloudFront will return an HTTP 502 status code (Bad Gateway) to the client. For more information, see DNS error (`NonS3OriginDnsError`).

If your function logic modifies the origin domain name, CloudFront will perform another DNS resolution on the updated domain name after the function has finished executing.

For more information about managing DNS failover, see Configuring DNS failover in the *Amazon Route 53 Developer Guide*.

HTTP status codes

Lambda@Edge functions for viewer response events cannot modify the HTTP status code of the response, regardless of whether the response came from the origin or the CloudFront cache.

Lambda function version

You must use a numbered version of the Lambda function, not `$LATEST` or aliases.

Lambda Region

The Lambda function must be in the US East (N. Virginia) Region.

Lambda role permissions

The IAM execution role associated with the Lambda function must allow the service principals `lambda.amazonaws.com` and `edgelambda.amazonaws.com` to assume the role. For more information, see Set up IAM permissions and roles for Lambda@Edge.

Lambda features

The following Lambda features are not supported by Lambda@Edge:

- Lambda runtime management configurations other than **Auto** (default)
- Configuration of your Lambda function to access resources inside your VPC

- Lambda function dead letter queues
- Lambda environment variables (except for reserved environment variables, which are automatically supported)
- Lambda functions with Managing AWS Lambda dependencies with layers
- Using AWS X-Ray
- Lambda provisioned concurrency

> **Note**
>
> Lambda@Edge functions share the same Regional concurrency capabilities as all Lambda functions. For more information, see Quotas on Lambda@Edge.

- Create a Lambda function using a container image
- Lambda functions that use the arm64 architecture
- Lambda functions with more than 512 MB of ephemeral storage
- Using a customer managed key to encrypt your .zip deployment packages
- Self-managed Amazon S3 code storage

Supported runtimes

Lambda@Edge supports the latest versions of Node.js and Python runtimes. For a list of supported versions and their future deprecation dates, see Supported runtimes in the *AWS Lambda Developer Guide*.

> **Tip**
>
> - As a best practice, use the latest versions of the provided runtimes for performance improvements and new features.
> - You can't create or update functions with deprecated versions of Node.js. You can only associate existing functions with these versions with CloudFront distributions. Functions with these versions that are associated with distributions will continue to run. However, we recommend that you move your function to newer versions of Node.js. For more information, see Runtime deprecation policy in the *AWS Lambda Developer Guide* and the Node.js release schedule on GitHub.

CloudFront headers

Lambda@Edge functions can read, edit, remove, or add any of the CloudFront headers listed in Add CloudFront request headers.

Notes

- If you want CloudFront to add these headers, you must configure CloudFront to add them by using a cache policy or origin request policy.
- CloudFront adds the headers *after* the viewer request event, which means the headers aren't available to Lambda@Edge functions in a viewer request. The headers are only available to Lambda@Edge functions in an origin request and origin response.
- If the viewer request includes headers that have these names, and you configured CloudFront to add these headers using a cache policy or origin request policy, then CloudFront overwrites the header values that were in the viewer request. Viewer-facing functions see the header value from the viewer request, while origin-facing functions see the header value that CloudFront added.
- If a viewer request function adds the `CloudFront-Viewer-Country` header, it fails validation and CloudFront returns HTTP status code 502 (Bad Gateway) to the viewer.

Restrictions on the request body with the include body option

When you choose the **Include Body** option to expose the request body to your Lambda@Edge function, the following information and size limits apply to the portions of the body that are exposed or replaced.

- CloudFront always base64 encodes the request body before exposing it to Lambda@Edge.
- If the request body is large, CloudFront truncates it before exposing it to Lambda@Edge, as follows:
  - For viewer request events, the body is truncated at 40 KB.
  - For origin request events, the body is truncated at 1 MB.
- If you access the request body as read-only, CloudFront sends the full original request body to the origin.
- If your Lambda@Edge function replaces the request body, the following size limits apply to the body that the function returns:

- If the Lambda@Edge function returns the body as plain text:
  - For viewer request events, the body limit is 40 KB.
  - For origin request events, the body limit is 1 MB.
- If the Lambda@Edge function returns the body as base64 encoded text:
  - For viewer request events, the body limit is 53.2 KB.
  - For origin request events, the body limit is 1.33 MB.

> **Note**
>
> If your Lambda@Edge function returns a body that exceeds these limits, your request will fail with an HTTP 502 status code (Lambda validation error). We recommend that you update your Lambda@Edge function so that the body doesn't exceed these limits.

Response timeout and keep-alive timeout (custom origins only)

If you're using Lambda@Edge functions to set the response timeout or keep-alive timeout for your distribution origins, verify that you're specifying a value that your origin can support. For more information, see Response and keep-alive timeout quotas.

Reports, metrics, and logs

CloudFront provides several options for reporting, monitoring, and logging your CloudFront resources. You can complete the following tasks:

- View and download reports to see usage and activity for your CloudFront distributions, including billing reports, cache statistics, popular content, and top referrers.
- Monitor and track CloudFront, including your edge computing functions, directly in the CloudFront console or by using Amazon CloudWatch. CloudFront sends metrics to CloudWatch for distributions and edge functions, both Lambda@Edge and CloudFront Functions.
- View logs for the viewer requests that your CloudFront distributions receive with standard logs or real-time access logs. In addition to viewer request logs, you can use CloudWatch Logs to get logs for your edge functions, both Lambda@Edge and CloudFront Functions. You can also use AWS CloudTrail to get logs of the CloudFront API activity in your AWS account.
- Track configuration changes to your CloudFront resources using AWS Config.

For more information about these options, see the following topics.

Topics

- AWS billing and usage reports for CloudFront
- View CloudFront reports in the console
- Monitor CloudFront metrics with Amazon CloudWatch
- CloudFront and edge function logging
- Track configuration changes with AWS Config

AWS billing and usage reports for CloudFront

AWS provides two usage reports for CloudFront:

- The AWS billing report is a high-level view of all activity for AWS services that you're using, including CloudFront.
- The AWS usage report is a summary of activity for a specific service, aggregated by hour, day, or month. It also includes usage charts that provide a graphical representation of your CloudFront usage.

> **Note**
>
> Like other AWS services, CloudFront charges you for only what you use. For more information, see CloudFront pricing.

Topics

- View the AWS billing report for CloudFront
- View the AWS usage report for CloudFront
- Interpret your AWS bill and usage reports for CloudFront

View the AWS billing report for CloudFront

You can view a summary of your AWS usage and charges, listed by service, on the **Bills** page in the AWS Billing and Cost Management console.

To view the AWS billing report

1. Sign in to the AWS Management Console and open the AWS Billing and Cost Management console at https://eusc-de-east-1.console.amazonaws-eusc.eu/costmanagement/.
2. In the navigation pane, choose **Bills**.
3. Choose a **Billing period** (for example, August 2023).
4. On the **Charges by service** tab, choose **CloudFront**, and then expand **Global** or the AWS Region name.
5. To download a detailed billing report in CSV format, choose **Download all to CSV**.

For more information about your AWS bill, see Viewing your bill in the *AWS Billing User Guide*.

The billing report includes the following values that apply to CloudFront:

- **ProductCode** – `AmazonCloudFront`
- **UsageType** – One of the following values:
  - A code that identifies the type of data transfer
  - `Invalidations`
  - `Executions-CloudFrontFunctions`

  - `NetworkCalls-CloudFrontFunctions`
  - `KeyValueStore-APIOperations`
  - `KeyValueStore-EdgeReads`
  - `RealTimeLog-KinesisDataStream`
  - `SSL-Cert-Custom`
- **ItemDescription** – A description of the billing rate for the **UsageType**.
- **UsageStartDate** and **UsageEndDate** – The day that the usage applies to, in Coordinated Universal Time (UTC).
- **UsageQuantity** – One of the following values:
  - The number of requests during the specified time period
  - The amount of data transferred in gigabytes
  - The number of objects invalidated
  - The sum of the prorated months that you had SSL certificates associated with enabled CloudFront distributions. For example, if you have one certificate associated with an enabled distribution for an entire month and another certificate associated with an enabled distribution for half of the month, this value will be 1.5.

View the AWS usage report for CloudFront

AWS provides a CloudFront usage report that is more detailed than the billing report but less detailed than CloudFront access logs. The usage report provides aggregate usage data by hour, day, or month, and it lists operations by region and usage type, such as data transferred out of the Australia region.

To view the AWS usage report

1. Sign in to the AWS Management Console and open the AWS Billing and Cost Management console at https://eusc-de-east-1.console.amazonaws-eusc.eu/costmanagement/.
2. In the navigation pane, choose **Cost Explorer**.
3. On the **New cost and usage report** page, in the **Report parameters** pane, choose a date range and granularity for the report.
4. Under **Filters**, **Service**, select **CloudFront**.
5. Select the **Usage type**.

6. Under **Cost and usage breakdown**, choose **Download as CSV**.

For more information about the AWS usage report, see AWS Usage Report in the *AWS Data Exports User Guide*.

The CloudFront usage report includes the following values:

- **Service** – `AmazonCloudFront`
- **Operation** – HTTP method. Values include `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST`, and `PUT`.
- **UsageType** – One of the following values:
  - A code that identifies the type of data transfer
  - `Invalidations`
  - `Executions-CloudFrontFunctions`
  - `NetworkCalls-CloudFrontFunctions`
  - `KeyValueStore-APIOperations`
  - `KeyValueStore-EdgeReads`
  - `RealTimeLog-KinesisDataStream`
  - `SSL-Cert-Custom`
- **Resource** – Either the ID of the CloudFront distribution associated with the usage or the certificate ID of an SSL certificate that you have associated with a CloudFront distribution.
- **StartTime/EndTime** – The day that the usage applies to, in Coordinated Universal Time (UTC).
- **UsageValue** – 1) The number of requests during the specified time period or 2) the amount of data transferred in bytes.

If you're using Amazon S3 as the origin for CloudFront, consider running the usage report for Amazon S3, too. However, if you use Amazon S3 for purposes other than as an origin for your CloudFront distributions, it might not be clear what portion applies to your CloudFront usage.

> **Tip**
>
> For detailed information about every request that CloudFront receives for your objects, turn on CloudFront access logs for your distribution. For more information, see the section called "Access logs (standard logs)".

For more information about understanding the CloudFront charges and usage types on your reports, see the section called “Interpret your AWS bill and usage reports for CloudFront”.

Interpret your AWS bill and usage reports for CloudFront

After you have the billing report and the usage report, you can use this topic to understand how to interpret each CloudFront charge that appears on your bill and the corresponding usage type for each charge. This topic includes the codes and AWS Region abbreviations that can appear on both reports.

Most codes in both columns include a two-letter abbreviation that indicates the location of the activity. In the following table, *`region`* in a code is replaced in your AWS bill and in the usage report by one of the following two-letter abbreviations:

- **AP:** Hong Kong, Philippines, South Korea, and Singapore (Asia Pacific)
- **AU:** Australia
- **CA:** Canada
- **EU:** Europe and Israel
- **IN:** India
- **JP:** Japan
- **ME:** Middle East
- **SA:** South America
- **US:** United States
- **ZA:** South Africa

For more information about pricing by AWS Region, see Amazon CloudFront pricing.

Notes

- This table doesn't include charges for transferring your objects from an Amazon S3 bucket to CloudFront edge locations. These charges, if any, appear in the **AWS Data Transfer** portion of your AWS bill.
- The first column lists charges that appear in your AWS bill report and explains what each means.

- The second column lists items that appear in the AWS usage report and shows the correlation between bill charges and usage report items.

CloudFront charges in your AWS bill	Values in the UsageType column in the AWS usage report
region-DataTransfer-Out-Bytes Total bytes served from CloudFront edge locations in *region* in response to user GET and HEAD requests.	*region*-Out-Bytes-HTTP-Static: Bytes served by using HTTP for objects with TTL ≥ 3,600 seconds. *region*-Out-Bytes-HTTPS-Static: Bytes served by using HTTPS for objects with TTL ≥ 3,600 seconds. *region*-Out-Bytes-HTTP-Dynamic: Bytes served by using HTTP for objects with TTL < 3,600 seconds. *region*-Out-Bytes-HTTPS-Dynamic: Bytes served by using HTTPS for objects with TTL < 3,600 seconds. *region*-Out-Bytes-HTTP-Proxy: Bytes returned from CloudFront to viewers by using HTTP in response to DELETE, OPTIONS, PATCH, POST, and PUT requests. *region*-Out-Bytes-HTTPS-Proxy: Bytes returned from CloudFront to viewers by using HTTPS in response to DELETE, OPTIONS, PATCH, POST, and PUT requests.

CloudFront charges in your AWS bill	Values in the UsageType column in the AWS usage report
	This includes bytes returned from CloudFront to viewers by using gRPC.
region-DataTransfer-Out-OBytes Total bytes transferred from CloudFront edge locations to your origin or edge function in response to DELETE, OPTIONS, PATCH, POST, and PUT requests. The charges include data transfer for WebSocket data from client to server.	*region*-Out-OBytes-HTTP-Proxy Total bytes transferred by using HTTP from CloudFront edge locations to your origin or edge function in response to DELETE, OPTIONS, PATCH, POST, and PUT requests. *region*-Out-OBytes-HTTPS-Proxy Total bytes transferred by using HTTPS from CloudFront edge locations to your origin or edge function in response to DELETE, OPTIONS, PATCH, POST, and PUT requests. This includes bytes transferred by using gRPC from CloudFront edge locations to your origin or CloudFront Functions.
region-Requests-Tier1 Number of HTTP GET and HEAD requests.	*region*-Requests-HTTP-Static Number of HTTP GET and HEAD requests served for objects with TTL ≥ 3,600 seconds. *region*-Requests-HTTP-Dynamic Number of HTTP GET and HEAD requests served for objects with TTL < 3,600 seconds.

CloudFront charges in your AWS bill	Values in the UsageType column in the AWS usage report
region-Requests-Tier2-HTTPS Number of HTTPS GET and HEAD requests.	*region*-Requests-HTTPS-Static Number of HTTPS GET and HEAD requests served for objects with TTL ≥ 3,600 seconds. *region*-Requests-HTTPS-Dynamic Number of HTTPS GET and HEAD requests served for objects with TTL < 3,600 seconds.
region-Requests-HTTP-Proxy Number of HTTP DELETE, OPTIONS, PATCH, POST, and PUT requests that CloudFront forwards to your origin or edge function. Also includes the number of HTTP WebSocket requests (GET requests with the Upgrade: websocket header) that CloudFront forwards to your origin or edge function.	*region*-Requests-HTTP-Proxy Same as the corresponding item in your CloudFront bill.
region-Requests-HTTPS-Proxy Number of HTTPS DELETE, OPTIONS, PATCH, POST, and PUT requests that CloudFront forwards to your origin or edge function. Also includes the following request types: • The number of HTTPS WebSocket requests (GET requests with the Upgrade: websocket header) that CloudFront forwards to your origin or edge function. • The number of HTTPS gRPC requests.	*region*-Requests-HTTPS-Proxy Same as the corresponding item in your CloudFront bill.

CloudFront charges in your AWS bill	Values in the UsageType column in the AWS usage report
region-Requests-HTTPS-Proxy-FLE Number of HTTPS DELETE, OPTIONS, PATCH, and POST requests processed with field-level encryption that CloudFront forwards to your origin or edge function.	*region*-Requests-HTTPS-Proxy-FLE Same as the corresponding item in your CloudFront bill.
region-Bytes-OriginShield Total bytes transferred from the origin to any regional edge cache, including the regional edge cache that is enabled as Origin Shield.	*region*-Bytes-OriginShield Same as the corresponding item in your CloudFront bill.
region-OBytes-OriginShield Total bytes transferred to the origin from any regional edge cache, including the regional edge cache that is enabled as Origin Shield.	*region*-OBytes-OriginShield Same as the corresponding item in your CloudFront bill.
region-Requests-OriginShield Number of requests that go to Origin Shield as an incremental layer. For dynamic (non-cacheable) requests that are proxied to the origin, Origin Shield is always an incremental layer. For cacheable requests, Origin Shield is sometimes an incremental layer. For more information, see the section called "Estimate Origin Shield costs".	*region*-Requests-OriginShield Same as the corresponding item in your CloudFront bill.

CloudFront charges in your AWS bill	Values in the UsageType column in the AWS usage report
Invalidations The charge for invalidating objects (removing the objects from CloudFront edge locations). For more information, see Pay for file invalidation.	**Invalidations** Same as the corresponding item in your CloudFront bill.
SSL-Cert-Custom The charge for using an SSL certificate with a CloudFront alternate domain name such as example.com instead of using the default CloudFront SSL certificate and the domain name that CloudFront assigned to your distribution.	**SSL-Cert-Custom** Same as the corresponding item in your CloudFront bill.
RealTimeLog-KinesisDataStream The charge for the number of lines generated for real-time access logs.	**RealTimeLog-KinesisDataStream** Same as the corresponding item in your CloudFront bill.
Executions-CloudFrontFunctions The charge for the number of CloudFront Functions invocations.	**Executions-CloudFrontFunctions** Same as the corresponding item in your CloudFront bill.
NetworkCalls-CloudFrontFunctions The charge for the number of CloudFront Functions network calls.	**NetworkCalls-CloudFrontFunctions** Same as the corresponding item in your CloudFront bill.
***region*-Lambda-Edge-Request** The charge for the number of Lambda@Edge function invocations.	***region*-Lambda-Edge-Request** Same as the corresponding item in your CloudFront bill.

CloudFront charges in your AWS bill	Values in the UsageType column in the AWS usage report
***region*-Lambda-Edge-GB-Second** The charge for the duration from when your Lambda@Edge function is invoked to when it returns or terminates.	***region*-Lambda-Edge-GB-Second** Same as the corresponding item in your CloudFront bill.
KeyValueStore-EdgeReads The charge for the number of read calls to the CloudFront KeyValueStore methods, `get()`, `exists()`, and `meta()`. For more informati on, see Helper methods for key value stores.	**KeyValueStore-EdgeReads** Same as the corresponding item in your CloudFront bill.
KeyValueStore-APIOperations The charge for the number of calls to the CloudFront KeyValueStore API.	**KeyValueStore-APIOperations** Same as the corresponding item in your CloudFront bill.

View CloudFront reports in the console

Each report provides detailed information and visualizations, so that you can optimize content delivery, identify performance bottlenecks, and make data-driven decisions. Whether you need to monitor cache efficiency, analyze traffic patterns, or understand your viewers better, you can use these reports to effectively monitor and analyze your CloudFront distributions.

You can view the following reports of your CloudFront activity in the console:

Topics

- View CloudFront cache statistics reports
- View CloudFront popular objects reports
- View CloudFront top referrers reports
- View CloudFront usage reports
- View CloudFront viewers reports

Most of these reports are based on the data in CloudFront access logs, which contain detailed information about every user request that CloudFront receives. You don't need to enable access logs to view the reports. For more information, see Access logs (standard logs).

View CloudFront cache statistics reports

The Amazon CloudFront cache statistics report shows the following information:

- **Total requests** – The total number of requests for all HTTP status codes (for example, 200 or 404) and all methods (for example, GET, HEAD, or POST)
- **Percentage of viewer requests by result type** – Hits, misses, and errors as a percentage of total viewer requests for the selected CloudFront distribution
- **Bytes transferred to viewers** – Total bytes and bytes from misses
- **HTTP status codes** – Viewer requests by HTTP status code
- **Percentage of GET requests that didn't finish downloading** – Viewer GET requests that didn't finish downloading the requested object as a percentage of total requests

Data for these statistics is drawn from the same source as CloudFront access logs. However, you don't need to enable access logging to view cache statistics.

You can display charts for a specified date range in the last 60 days, with data points every hour or every day. You can usually view data about requests that CloudFront received as recently as an hour ago, but data can occasionally be delayed by as much as 24 hours.

Topics

- View CloudFront cache statistics reports in the console
- Download data in CSV format
- How cache statistics charts are related to data in the CloudFront standard logs (access logs)

View CloudFront cache statistics reports in the console

You can view the CloudFront cache statistics report in the console.

To view the CloudFront cache statistics report

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.

2. In the navigation pane, choose **Cache Statistics**.
3. In the **CloudFront Cache Statistics Reports** pane, for **Start Date** and **End Date**, select the date range for which you want to display cache statistics charts. Available ranges depend on the value that you select for **Granularity**:

   - **Daily** – To display charts with one data point per day, select any date range in the previous 60 days.
   - **Hourly** – To display charts with one data point every hour, select any date range of up to 14 days within the previous 60 days.

   Dates and times are in Coordinated Universal Time (UTC).
4. For **Granularity**, specify whether to display one data point per day or one data point per hour in the charts. If you specify a date range greater than 14 days, the option to specify one data point per hour is not available.
5. For **Viewer Location**, choose the continent from which viewer requests originated, or choose **All Locations**. Cache statistics charts include data for requests that CloudFront received from the specified location.
6. In the **Distribution** list, select the distributions for which you want to display data in the usage charts:

   - **An individual distribution** – The charts display data for the selected CloudFront distribution. The **Distribution** list displays the distribution ID and alternate domain names (CNAMEs) for the distribution, if any. If a distribution doesn't have alternate domain names, the list includes origin domain names for the distribution.
   - **All distributions** – The charts display summed data for all distributions that are associated with the current AWS account, excluding distributions that you have deleted.
7. Choose **Update**.

Tip

- To view data for a daily or hourly data point within a chart, hover over the data point.
- For charts that show data transferred, you can change the vertical scale to gigabytes, megabytes, or kilobytes.

Download data in CSV format

You can download the cache statistics report in CSV format. This section explains how to download the report and describes the values in the report.

To download the cache statistics report in CSV format

1. While viewing the cache statistics report, choose **CSV**.
2. In the **Opening *file name*** dialog box, choose whether to open or save the file.

Information about the report

The first few rows of the report include the following information:

Version

The version of the format for this CSV file.

Report

The name of the report.

DistributionID

The ID of the distribution that you ran the report for, or ALL if you ran the report for all distributions.

StartDateUTC

The beginning of the date range for which you ran the report, in Coordinated Universal Time (UTC).

EndDateUTC

The end of the date range for which you ran the report, in Coordinated Universal Time (UTC).

GeneratedTimeUTC

The date and time on which you ran the report, in Coordinated Universal Time (UTC).

Granularity

Whether each row in the report represents one hour or one day.

ViewerLocation

The continent that viewer requests originated from, or ALL, if you chose to download the report for all locations.

Data in the cache statistics report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or ALL if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

ViewerLocation

The continent that viewer requests originated from, or ALL, if you chose to download the report for all locations.

TimeBucket

The hour or the day that data applies to, in Coordinated Universal Time (UTC).

RequestCount

The total number of requests for all HTTP status codes (for example, 200 or 404) and all methods (for example, GET, HEAD, or POST).

HitCount

The number of viewer requests for which the object is served from a CloudFront edge cache.

MissCount

The number of viewer requests for which the object isn't currently in an edge cache, so CloudFront must get the object from your origin.

ErrorCount

The number of viewer requests that resulted in an error, so CloudFront didn't serve the object.

IncompleteDownloadCount

The number of viewer requests for which the viewer started but didn't finish downloading the object.

HTTP2xx

The number of viewer requests for which the HTTP status code was a 2xx value (succeeded).

HTTP3xx

The number of viewer requests for which the HTTP status code was a 3xx value (additional action is required).

HTTP4xx

The number of viewer requests for which the HTTP status code was a 4xx value (client error).

HTTP5xx

The number of viewer requests for which the HTTP status code was a 5xx value (server error).

TotalBytes

The total number of bytes served to viewers by CloudFront in response to all requests for all HTTP methods.

BytesFromMisses

The number of bytes served to viewers for objects that were not in the edge cache at the time of the request. This value is a good approximation of bytes transferred from your origin to CloudFront edge caches. However, it excludes requests for objects that are already in the edge cache but that have expired.

How cache statistics charts are related to data in the CloudFront standard logs (access logs)

The following table shows how cache statistics charts in the CloudFront console correspond with values in CloudFront access logs. For more information about CloudFront access logs, see Access logs (standard logs).

Total requests

This chart shows the total number of requests for all HTTP status codes (for example, 200 or 404) and all methods (for example, `GET`, `HEAD`, or `POST`). Total requests shown in this chart equal the total number of requests in the access log files for the same time period.

Percentage of viewer requests by result type

This chart shows hits, misses, and errors as a percentage of total viewer requests for the selected CloudFront distribution:

- **Hit** – A viewer request for which the object is served from a CloudFront edge cache. In access logs, these are requests for which the value of `x-edge-response-result-type` is `Hit`.
- **Miss** – A viewer request for which the object isn't currently in an edge cache, so CloudFront must get the object from your origin. In access logs, these are requests for which the value of `x-edge-response-result-type` is `Miss`.
- **Error** – A viewer request that resulted in an error, so CloudFront didn't serve the object. In access logs, these are requests for which the value of `x-edge-response-result-type` is `Error`, `LimitExceeded`, or `CapacityExceeded`.

The chart does not include refresh hits—requests for objects that are in the edge cache but that have expired. In access logs, refresh hits are requests for which the value of `x-edge-response-result-type` is `RefreshHit`.

Bytes transferred to viewers

This chart shows two values:

- **Total bytes** – The total number of bytes served to viewers by CloudFront in response to all requests for all HTTP methods. In CloudFront access logs, **Total Bytes** is the sum of the values in the `sc-bytes` column for all of the requests during the same time period.
- **Bytes from misses** – The number of bytes served to viewers for objects that were not in the edge cache at the time of the request. In CloudFront access logs, **bytes from misses** is the sum of the values in the `sc-bytes` column for requests for which the value of `x-edge-result-type` is `Miss`. This value is a good approximation of bytes transferred from your origin to CloudFront edge caches. However, it excludes requests for objects that are already in the edge cache but that have expired.

HTTP status codes

This chart shows viewer requests by HTTP status code. In CloudFront access logs, status codes appear in the `sc-status` column:

- **2xx** – The request succeeded.
- **3xx** – Additional action is required. For example, 301 (Moved Permanently) means that the requested object has moved to a different location.
- **4xx** – The client apparently made an error. For example, 404 (Not Found) means that the client requested an object that could not be found.
- **5xx** – The origin server didn't fill the request. For example, 503 (Service Unavailable) means that the origin server is currently unavailable.

Percentage of GET requests that didn't finish downloading

This chart shows viewer `GET` requests that didn't finish downloading the requested object as a percentage of total requests. Typically, downloading an object doesn't complete because the viewer canceled the download, for example, by choosing a different link or by closing the browser. In CloudFront access logs, these requests have a value of `200` in the `sc-status` column and a value of `Error` in the `x-edge-result-type` column.

View CloudFront popular objects reports

View the Amazon CloudFront popular objects report to see the 50 most popular objects for a distribution during a specified date range in the previous 60 days. You can also view statistics about those objects, including the following:

- Number of requests for the object
- Number of hits and misses
- Hit ratio
- Number of bytes served for misses
- Total bytes served
- Number of incomplete downloads
- Number of requests by HTTP status code (2xx, 3xx, 4xx, and 5xx)

Data for these statistics is drawn from the same source as CloudFront access logs. However, you don't need to enable access logging to view popular objects.

Topics

- View CloudFront popular objects reports in the console

- How CloudFront calculates popular objects statistics
- Download data in CSV format
- How data in the popular objects report is related to data in the CloudFront standard logs (access logs)

View CloudFront popular objects reports in the console

You can view the CloudFront popular objects report in the console.

To view popular objects for a CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Popular Objects**.
3. In the **CloudFront Popular Objects Report** pane, for **Start Date** and **End Date**, select the date range for which you want to display a list of popular objects. You can choose any date range in the previous 60 days.

   Dates and times are in Coordinated Universal Time (UTC).
4. In the **Distribution** list, select the distribution for which you want to display a list of popular objects.
5. Choose **Update**.

How CloudFront calculates popular objects statistics

To get an accurate count of the top 50 objects in your distribution, CloudFront counts the requests for all of your objects in 10-minute intervals beginning at midnight and keeps a running total of the top 150 objects for the next 24 hours. (CloudFront also retains daily totals for the top 150 objects for 60 days.)

Near the bottom of the list, objects constantly rise onto or drop off the list, so the totals for those objects are approximations. The 50 objects at the top of the list of 150 objects might rise and fall within the list, but they rarely drop off the list altogether, so the totals for those objects are more reliable.

When an object drops off the list of the top 150 objects and then rises onto the list again over the course of a day, CloudFront adds an estimated number of requests for the period that the object

was missing from the list. The estimate is based on the number of requests received by whichever object was at the bottom of the list during that time period.

If the object rises into the top 50 objects later in the day, the estimates of the number of requests that CloudFront received while the object was out of the top 150 objects usually cause the number of requests in the popular objects report to exceed the number of requests that appear in the access logs for that object.

Download data in CSV format

You can download the popular objects report in CSV format. This section explains how to download the report and describes the values in the report.

To download the popular objects report in CSV format

1. While viewing the popular objects report, choose **CSV**.
2. In the **Opening *file name*** dialog box, choose whether to open or save the file.

Information about the report

The first few rows of the report include the following information:

Version

The version of the format for this CSV file.

Report

The name of the report.

DistributionID

The ID of the distribution that you ran the report for.

StartDateUTC

The beginning of the date range for which you ran the report, in Coordinated Universal Time (UTC).

EndDateUTC

The end of the date range for which you ran the report, in Coordinated Universal Time (UTC).

GeneratedTimeUTC

The date and time on which you ran the report, in Coordinated Universal Time (UTC).

Data in the popular objects report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

Object

The last 500 characters of the URL for the object.

RequestCount

The total number of requests for this object.

HitCount

The number of viewer requests for which the object is served from a CloudFront edge cache.

MissCount

The number of viewer requests for which the object isn't currently in an edge cache, so CloudFront must get the object from your origin.

HitCountPct

The value of `HitCount` as a percentage of the value of `RequestCount`.

BytesFromMisses

The number of bytes served to viewers for this object when the object was not in the edge cache at the time of the request.

TotalBytes

The total number of bytes served to viewers by CloudFront for this object in response to all requests for all HTTP methods.

IncompleteDownloadCount

The number of viewer requests for this object for which the viewer started but didn't finish downloading the object.

HTTP2xx

The number of viewer requests for which the HTTP status code was a 2xx value (succeeded).

HTTP3xx

The number of viewer requests for which the HTTP status code was a 3xx value (additional action is required).

HTTP4xx

The number of viewer requests for which the HTTP status code was a 4xx value (client error).

HTTP5xx

The number of viewer requests for which the HTTP status code was a 5xx value (server error).

How data in the popular objects report is related to data in the CloudFront standard logs (access logs)

The following list shows how values in the popular objects report in the CloudFront console correspond with values in CloudFront access logs. For more information about CloudFront access logs, see Access logs (standard logs).

URL

The last 500 characters of the URL that viewers use to access the object.

Requests

The total number of requests for the object. This value generally corresponds closely with the number of `GET` requests for the object in CloudFront access logs.

Hits

The number of viewer requests for which the object was served from a CloudFront edge cache. In access logs, these are requests for which the value of `x-edge-response-result-type` is `Hit`.

Misses

The number of viewer requests for which the object wasn't in an edge cache, so CloudFront retrieved the object from your origin. In access logs, these are requests for which the value of `x-edge-response-result-type` is `Miss`.

Hit ratio

The value of the **Hits** column as a percentage of the value of the **Requests** column.

Bytes from misses

The number of bytes served to viewers for objects that were not in the edge cache at the time of the request. In CloudFront access logs, **bytes from misses** is the sum of the values in the `sc-bytes` column for requests for which the value of `x-edge-result-type` is `Miss`.

Total bytes

The total number of bytes that CloudFront served to viewers in response to all requests for the object for all HTTP methods. In CloudFront access logs, **total bytes** is the sum of the values in the `sc-bytes` column for all of the requests during the same time period.

Incomplete downloads

The number of viewer requests that did not finish downloading the requested object. Typically, the reason that a download doesn't complete is that the viewer canceled it, for example, by choosing a different link or by closing the browser. In CloudFront access logs, these requests have a value of `200` in the `sc-status` column and a value of `Error` in the `x-edge-result-type` column.

2xx

The number of requests for which the HTTP status code is `2xx`, `Successful`. In CloudFront access logs, status codes appear in the `sc-status` column.

3xx

The number of requests for which the HTTP status code is `3xx`, `Redirection`. `3xx` status codes indicate that additional action is required. For example, 301 (Moved Permanently) means that the requested object has moved to a different location.

4xx

The number of requests for which the HTTP status code is `4xx`, `Client Error`. `4xx` status codes indicate that the client apparently made an error. For example, 404 (Not Found) means that the client requested an object that could not be found.

5xx

The number of requests for which the HTTP status code is `5xx`, `Server Error`. `5xx` status codes indicate that the origin server didn't fill the request. For example, 503 (Service Unavailable) means that the origin server is currently unavailable.

View CloudFront top referrers reports

The CloudFront top referrers report includes the following for any date range in the previous 60 days:

- Top 25 referrers (domains of websites that originated the most HTTP and HTTPS requests for objects that CloudFront is distributing for your distribution)
- Number of requests from a referrer
- Number of requests from a referrer as a percentage of the total number of requests during the specified period

Data for the top referrers report is drawn from the same source as CloudFront access logs. However, you don't need to enable access logging to view the top referrers.

Top referrers can be search engines, other websites that link directly to your objects, or your own website. For example, if `https://example.com/index.html` links to 10 graphics, `example.com` is the referrer for all 10 graphics.

> **Note**
>
> If a user enters a URL directly into the address line of a browser, there is no referrer for the requested object.

Topics

- View CloudFront top referrers reports in the console
- How CloudFront calculates top referrers statistics
- Download data in CSV format
- How data in the top referrers report is related to data in the CloudFront standard logs (access logs)

View CloudFront top referrers reports in the console

You can view the CloudFront top referrers report in the console.

To view top referrers for a CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Top Referrers**.
3. In the **CloudFront Top Referrers Report** pane, for **Start Date** and **End Date**, select the date range for which you want to display a list of top referrers.

   Dates and times are in Coordinated Universal Time (UTC).
4. In the **Distribution** list, select the distribution for which you want to display a list of top referrers.
5. Choose **Update**.

How CloudFront calculates top referrers statistics

To get an accurate count of the top 25 referrers, CloudFront counts the requests for all of your objects in 10-minute intervals and keeps a running total of the top 75 referrers. Near the bottom of the list, referrers constantly rise onto or drop off the list, so the totals for those referrers are approximations.

The 25 referrers at the top of the list of 75 referrers might rise and fall within the list, but they rarely drop off of the list altogether, so the totals for those referrers typically are more reliable.

Download data in CSV format

You can download the top referrers report in CSV format. This section explains how to download the report and describes the values in the report.

To download the top referrers report in CSV format

1. While viewing the Top Referrers report, choose **CSV**.
2. In the **Opening *file name*** dialog box, choose whether to open or save the file.

Information about the report

The first few rows of the report include the following information:

Version

The version of the format for this CSV file.

Report

The name of the report.

DistributionID

The ID of the distribution that you ran the report for, or ALL if you ran the report for all distributions.

StartDateUTC

The beginning of the date range for which you ran the report, in Coordinated Universal Time (UTC).

EndDateUTC

The end of the date range for which you ran the report, in Coordinated Universal Time (UTC).

GeneratedTimeUTC

The date and time on which you ran the report, in Coordinated Universal Time (UTC).

Data in the top referrers report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or ALL if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

Referrer

The domain name of the referrer.

RequestCount

The total number of requests from the domain name in the `Referrer` column.

RequestsPct

The number of requests submitted by the referrer as a percentage of the total number of requests during the specified period.

How data in the top referrers report is related to data in the CloudFront standard logs (access logs)

The following list shows how values in the Top Referrers report in the CloudFront console correspond with values in CloudFront access logs. For more information about CloudFront access logs, see Access logs (standard logs).

Referrer

The domain name of the referrer. In access logs, referrers are listed in the `cs(Referer)` column.

Request count

The total number of requests from the domain name in the **Referrer** column. This value generally corresponds closely with the number of `GET` requests from the referrer in CloudFront access logs.

Request %

The number of requests submitted by the referrer as a percentage of the total number of requests during the specified period. If you have more than 25 referrers, then you can't calculate **Request %** based on the data in this table because the **request count** column doesn't include all of the requests during the specified period.

View CloudFront usage reports

The CloudFront usage reports include the following information:

- **Number of requests** – Shows the total number of requests that CloudFront responds to from edge locations in the selected region during each time interval for the specified CloudFront distribution.

- **Data transferred by protocol** and **data transferred by destination** – Both show the total amount of data transferred from CloudFront edge locations in the selected region during each time interval for the specified CloudFront distribution. They separate the data differently, as follows:
  - **By protocol** – Separates the data by protocol: HTTP or HTTPS.
  - **By destination** – Separates the data by destination: to your viewers or to your origin.

The CloudFront usage report is based on the AWS usage report for CloudFront. This report doesn't require any additional configuration. For more information, see View the AWS usage report for CloudFront.

You can view reports for a specified date range in the last 60 days, with data points every hour or every day. You can usually view data about requests that CloudFront received as recently as four hours ago, but data can occasionally be delayed by as much as 24 hours.

For more information, see How the usage charts are related to data in the CloudFront usage report.

Topics

- View CloudFront usage reports in the console
- Download data in CSV format
- How the usage charts are related to data in the CloudFront usage report

View CloudFront usage reports in the console

You can view the CloudFront usage report in the console.

To view CloudFront usage reports

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Usage Reports**.
3. In the **CloudFront Usage Reports** pane, for **Start Date** and **End Date**, select the date range for which you want to display usage charts. Available ranges depend on the value that you select for **Granularity**:
   - **Daily** — To display charts with one data point per day, select any date range in the previous 60 days.

- **Hourly** — To display charts with one data point every hour, select any date range of up to 14 days within the previous 60 days.

Dates and times are in Coordinated Universal Time (UTC).

4. For **Granularity**, specify whether to display one data point per day or one data point per hour in the charts. If you specify a date range greater than 14 days, the option to specify one data point per hour is not available.
5. For **Billing Region**, choose the CloudFront billing region that has the data you want to view, or choose **All Regions**. Usage charts include data for requests that CloudFront processes in edge locations in the specified region. The region where CloudFront processes requests might or might not correspond with the location of your viewers.

   Select only Regions that are included in the price class for your distribution. Otherwise, the usage charts probably won't contain any data. For example, if you chose Price Class 200 for your distribution, the South America and Australia billing regions are not included, so CloudFront generally won't process your requests from those regions. For more information about price classes, see CloudFront pricing.
6. In the **Distribution** list, select the distributions for which you want to display data in the usage charts:

   - **An individual distribution** — The charts display data for the selected CloudFront distribution. The **Distribution** list displays the distribution ID and alternate domain names (CNAMEs) for the distribution, if any. If a distribution has no alternate domain names, the list includes origin domain names for the distribution.
   - **All distributions (excludes deleted)** — The charts display summed data for all distributions that are associated with the current AWS account, excluding distributions that you have deleted.
   - **All Deleted Distributions** — The charts display summed data for all distributions that are associated with the current AWS account and that were deleted in the last 60 days.
7. Choose **Update Graphs**.

Tip

- To view data for a daily or hourly data point within a chart, hover over the data point.

- For charts that show data transferred, note that you can change the vertical scale to gigabytes, megabytes, or kilobytes.

Download data in CSV format

You can download the usage report in CSV format. This section explains how to download the report and describes the values in the report.

To download the usage report in CSV format

1. While viewing the Usage report, choose **CSV**.
2. In the **Opening *file name*** dialog box, choose whether to open or save the file.

Information about the report

The first few rows of the report include the following information:

Version

The version of the format for this CSV file.

Report

The name of the report.

DistributionID

The ID of the distribution that you ran the report for, `ALL` if you ran the report for all distributions, or `ALL_DELETED` if you ran the report for all deleted distributions.

StartDateUTC

The beginning of the date range for which you ran the report, in Coordinated Universal Time (UTC).

EndDateUTC

The end of the date range for which you ran the report, in Coordinated Universal Time (UTC).

GeneratedTimeUTC

The date and time on which you ran the report, in Coordinated Universal Time (UTC).

Granularity

Whether each row in the report represents one hour or one day.

BillingRegion

The continent that viewer requests originated from, or ALL, if you chose to download the report for all billing regions.

Data in the usage report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, ALL if you ran the report for all distributions, or ALL_DELETED if you ran the report for all deleted distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

BillingRegion

The CloudFront billing region that you ran the report for, or ALL.

TimeBucket

The hour or the day that data applies to, in Coordinated Universal Time (UTC).

HTTP

The number of HTTP requests that CloudFront responded to from edge locations in the selected region during each time interval for the specified CloudFront distribution. Values include:

- The number of GET and HEAD requests, which cause CloudFront to transfer data to your viewers
- The number of DELETE, OPTIONS, PATCH, POST, and PUT requests, which cause CloudFront to transfer data to your origin

HTTPS

The number of HTTPS requests that CloudFront responded to from edge locations in the selected region during each time interval for the specified CloudFront distribution. Values include:

- The number of GET and HEAD requests, which cause CloudFront to transfer data to your viewers
- The number of DELETE, OPTIONS, PATCH, POST, and PUT requests, which cause CloudFront to transfer data to your origin

HTTPBytes

The total amount of data transferred over HTTP from CloudFront edge locations in the selected billing region during the time period for the specified CloudFront distribution. Values include:

- Data transferred from CloudFront to your viewers in response to GET and HEAD requests
- Data transferred from your viewers to CloudFront for DELETE, OPTIONS, PATCH, POST, and PUT requests
- Data transferred from CloudFront to your viewers in response to DELETE, OPTIONS, PATCH, POST, and PUT requests

HTTPSBytes

The total amount of data transferred over HTTPS from CloudFront edge locations in the selected billing region during the time period for the specified CloudFront distribution. Values include:

- Data transferred from CloudFront to your viewers in response to GET and HEAD requests
- Data transferred from your viewers to CloudFront for DELETE, OPTIONS, PATCH, POST, and PUT requests
- Data transferred from CloudFront to your viewers in response to DELETE, OPTIONS, PATCH, POST, and PUT requests

BytesIn

The total amount of data transferred from CloudFront to your origin for DELETE, OPTIONS, PATCH, POST, and PUT requests in the selected region during each time interval for the specified CloudFront distribution.

BytesOut

The total amount of data transferred over HTTP and HTTPS from CloudFront to your viewers in the selected region during each time interval for the specified CloudFront distribution. Values include:

- Data transferred from CloudFront to your viewers in response to `GET` and `HEAD` requests
- Data transferred from CloudFront to your viewers in response to `DELETE`, `OPTIONS`, `PATCH`, `POST`, and `PUT` requests

How the usage charts are related to data in the CloudFront usage report

The following list shows how the usage charts in the CloudFront console correspond with values in the **Usage Type** column in the CloudFront usage report.

Topics

- Number of requests
- Data transferred by protocol
- Data transferred by destination

Number of requests

This chart shows the total number of requests that CloudFront responds to from edge locations in the selected region during each time interval for the specified CloudFront distribution, separated by protocol (HTTP or HTTPS) and type (static, dynamic, or proxy).

Number of HTTP requests

- *`region`*-**Requests-HTTP-Static:** Number of HTTP `GET` and `HEAD` requests served for objects with TTL ≥ 3600 seconds
- *`region`*-**Requests-HTTP-Dynamic:** Number of HTTP `GET` and `HEAD` requests served for objects with TTL < 3600 seconds
- *`region`*-**Requests-HTTP-Proxy:** Number of HTTP `DELETE`, `OPTIONS`, `PATCH`, `POST`, and `PUT` requests that CloudFront forwards to your origin

Number of HTTPS requests

- *`region`*-**Requests-HTTPS-Static:** Number of HTTPS `GET` and `HEAD` requests served for objects with TTL ≥ 3600 seconds

- *region*-**Requests-HTTPS-Dynamic:** Number of HTTPS GET and HEAD requests served for objects with TTL < 3600 seconds
- *region*-**Requests-HTTPS-Proxy:** Number of HTTPS DELETE, OPTIONS, PATCH, POST, and PUT requests that CloudFront forwards to your origin

Data transferred by protocol

This chart shows the total amount of data transferred from CloudFront edge locations in the selected region during each time interval for the specified CloudFront distribution, separated by protocol (HTTP or HTTPS), type (static, dynamic, or proxy), and destination (viewers or origin).

Data transferred over HTTP

- *region*-**Out-Bytes-HTTP-Static:** Bytes served by using HTTP for objects with TTL ≥ 3600 seconds
- *region*-**Out-Bytes-HTTP-Dynamic:** Bytes served by using HTTP for objects with TTL < 3600 seconds
- *region*-**Out-Bytes-HTTP-Proxy:** Bytes returned from CloudFront to viewers by using HTTP in response to DELETE, OPTIONS, PATCH, POST, and PUT requests
- *region*-**Out-OBytes-HTTP-Proxy:** Total bytes transferred by using HTTP from CloudFront edge locations to your origin in response to DELETE, OPTIONS, PATCH, POST, and PUT requests

Data transferred over HTTPS

- *region*-**Out-Bytes-HTTPS-Static:** Bytes served by using HTTPS for objects with TTL ≥ 3600 seconds
- *region*-**Out-Bytes-HTTPS-Dynamic:** Bytes served by using HTTPS for objects with TTL < 3600 seconds
- *region*-**Out-Bytes-HTTPS-Proxy:** Bytes returned from CloudFront to viewers by using HTTPS in response to DELETE, OPTIONS, PATCH, POST, and PUT requests
- *region*-**Out-OBytes-HTTPS-Proxy:** Total bytes transferred by using HTTPS from CloudFront edge locations to your origin in response to DELETE, OPTIONS, PATCH, POST, and PUT requests

Data transferred by destination

This chart shows the total amount of data transferred from CloudFront edge locations in the selected region during each time interval for the specified CloudFront distribution, separated by destination (viewers or origin), protocol (HTTP or HTTPS), and type (static, dynamic, or proxy).

Data transferred from CloudFront to your viewers

- *region*-**Out-Bytes-HTTP-Static:** Bytes served by using HTTP for objects with TTL ≥ 3600 seconds
- *region*-**Out-Bytes-HTTPS-Static:** Bytes served by using HTTPS for objects with TTL ≥ 3600 seconds
- *region*-**Out-Bytes-HTTP-Dynamic:** Bytes served by using HTTP for objects with TTL < 3600 seconds
- *region*-**Out-Bytes-HTTPS-Dynamic:** Bytes served by using HTTPS for objects with TTL < 3600 seconds
- *region*-**Out-Bytes-HTTP-Proxy:** Bytes returned from CloudFront to viewers by using HTTP in response to `DELETE`, `OPTIONS`, `PATCH`, `POST`, and `PUT` requests
- *region*-**Out-Bytes-HTTPS-Proxy:** Bytes returned from CloudFront to viewers by using HTTPS in response to `DELETE`, `OPTIONS`, `PATCH`, `POST`, and `PUT` requests

Data transferred from CloudFront to your origin

- *region*-**Out-OBytes-HTTP-Proxy:** Total bytes transferred by using HTTP from CloudFront edge locations to your origin in response to `DELETE`, `OPTIONS`, `PATCH`, `POST`, and `PUT` requests
- *region*-**Out-OBytes-HTTPS-Proxy:** Total bytes transferred by using HTTPS from CloudFront edge locations to your origin in response to `DELETE`, `OPTIONS`, `PATCH`, `POST`, and `PUT` requests

View CloudFront viewers reports

The CloudFront viewers reports include the following information for any date range in the previous 60 days:

- **Devices** – The types of devices most frequently used to access your content (such as Desktop or Mobile)

- **Browsers** – The top 10 browsers most frequently used to access your content (such as Chrome or Firefox)
- **Operating systems** – The top 10 operating systems most frequently used when accessing your content (such as Linux, macOS, or Windows)
- **Locations** – The top 50 locations (countries or U.S. states/territories) of the viewers that most frequently access your content
  - You can also view locations with hourly data points for any date range of up to 14 days in the previous 60 days

> **Note**
>
> You don't need to enable access logging to see viewers charts and reports.

Topics

- View viewers charts and reports in the console
- Download data in CSV format
- Data included in the viewers reports
- How data in the locations report is related to data in the CloudFront standard logs (access logs)

View viewers charts and reports in the console

You can view CloudFront viewers charts and reports in the console.

To view CloudFront viewers charts and reports

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Viewers**.
3. In the **CloudFront Viewers** pane, for **Start Date** and **End Date**, select the date range for which you want to display viewer charts and reports.

   For the Locations chart, available ranges depend on the value that you select for **Granularity**:

   - **Daily** – To display charts with one data point per day, select any date range in the previous 60 days.

- **Hourly** – To display charts with one data point every hour, select any date range of up to 14 days within the previous 60 days.

Dates and times are in Coordinated Universal Time (UTC).

4. (Browsers and Operating Systems charts only) For **Grouping**, specify whether you want to group browsers and operating systems by name (Chrome, Firefox) or by name and version (Chrome 40.0, Firefox 35.0).
5. (Locations chart only) For **Granularity**, specify whether to display one data point per day or one data point per hour in the charts. If you specify a date range greater than 14 days, the option to specify one data point per hour is not available.
6. (Locations chart only) For **Details**, specify whether to display the top locations by countries or by U.S. states.
7. In the **Distribution** list, select the distribution for which you want to display data in the usage charts:

   - **An individual distribution** – The charts display data for the selected CloudFront distribution. The **Distribution** list displays the distribution ID and an alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.
   - **All distributions (excludes deleted)** – The charts display summed data for all distributions that are associated with the current AWS account, excluding distributions that you have deleted.
8. Choose **Update**.

To view data for a daily or hourly data point within a chart, hover over the data point.

Download data in CSV format

You can download each of the viewer reports in CSV format. This section explains how to download the reports and describes the values in the report.

To download the viewer reports in CSV format

1. While viewing the Viewer report, choose **CSV**.
2. Choose the data that you want to download, for example, **Devices** or **Devices Trends**.
3. In the **Opening *file name*** dialog box, choose whether to open or save the file.

Data included in the viewers reports

The first few rows of each report include the following information:

Version

The version of the format for this CSV file.

Report

The name of the report.

DistributionID

The ID of the distribution that you ran the report for, or ALL if you ran the report for all distributions.

StartDateUTC

The beginning of the date range for which you ran the report, in Coordinated Universal Time (UTC).

EndDateUTC

The end of the date range for which you ran the report, in Coordinated Universal Time (UTC).

GeneratedTimeUTC

The date and time on which you ran the report, in Coordinated Universal Time (UTC).

Grouping (browsers and operating systems reports only)

Whether the data is grouped by the name or by the name and version of the browser or operating system.

Granularity

Whether each row in the report represents one hour or one day.

Details (locations report only)

Whether requests are listed by country or by U.S. state.

The following topics describe the information in the different viewers reports.

Topics

- Devices report
- Device trends report
- Browsers report
- Browser trends report
- Operating systems report
- Operating system trends report
- Locations report
- Location trends report

Devices report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

Requests

The number of requests that CloudFront received from each type of device.

RequestsPct

The number of requests that CloudFront received from each type of device as a percentage of the total number of requests that CloudFront received from all devices.

Custom

Requests for which the value of the `User-Agent` HTTP header was not associated with one of the standard device types, for example, `Desktop` or `Mobile`.

Device trends report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

TimeBucket

The hour or the day that the data applies to, in Coordinated Universal Time (UTC).

Desktop

The number of requests that CloudFront received from desktop computers during the period.

Mobile

The number of requests that CloudFront received from mobile devices during the period. Mobile devices can include both tablets and mobile phones. If CloudFront can't determine whether a request originated from a mobile device or a tablet, it's counted in the `Mobile` column.

Smart-TV

The number of requests that CloudFront received from smart TVs during the period.

Tablet

The number of requests that CloudFront received from tablets during the period. If CloudFront can't determine whether a request originated from a mobile device or a tablet, it's counted in the `Mobile` column.

Unknown

Requests for which the value of the `User-Agent` HTTP header was not associated with one of the standard device types, for example, `Desktop` or `Mobile`.

Empty

The number of requests that CloudFront received that didn't include a value in the HTTP `User-Agent` header during the period.

Browsers report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

Group

The browser or the browser and version that CloudFront received requests from, depending on the value of `Grouping`. In addition to browser names, possible values include the following:

- **Bot/Crawler** – primarily requests from search engines that are indexing your content.
- **Empty** – requests for which the value of the `User-Agent` HTTP header was empty.
- **Other** – browsers that CloudFront identified but that aren't among the most popular. If `Bot/Crawler`, `Empty`, and/or `Unknown` don't appear among the first nine values, then they're also included in `Other`.
- **Unknown** – requests for which the value of the `User-Agent` HTTP header was not associated with a standard browser. Most requests in this category come from custom applications or scripts.

Requests

The number of requests that CloudFront received from each type of browser.

RequestsPct

The number of requests that CloudFront received from each type of browser as a percentage of the total number of requests that CloudFront received during the time period.

Browser trends report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

TimeBucket

The hour or the day that the data applies to, in Coordinated Universal Time (UTC).

(Browsers)

The remaining columns in the report list the browsers or the browsers and their versions, depending on the value of `Grouping`. In addition to browser names, possible values include the following:

- **Bot/Crawler** – primarily requests from search engines that are indexing your content.
- **Empty** – requests for which the value of the `User-Agent` HTTP header was empty.
- **Other** – browsers that CloudFront identified but that aren't among the most popular. If `Bot/Crawler`, `Empty`, and/or `Unknown` don't appear among the first nine values, then they're also included in `Other`.
- **Unknown** – requests for which the value of the `User-Agent` HTTP header was not associated with a standard browser. Most requests in this category come from custom applications or scripts.

Operating systems report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

Group

The operating system or the operating system and version that CloudFront received requests from, depending on the value of `Grouping`. In addition to operating system names, possible values include the following:

- **Bot/Crawler** – primarily requests from search engines that are indexing your content.
- **Empty** – requests for which the value of the `User-Agent` HTTP header was empty.
- **Other** – operating systems that CloudFront identified but that aren't among the most popular. If `Bot/Crawler`, `Empty`, and/or `Unknown` don't appear among the first nine values, then they're also included in `Other`.
- **Unknown** – requests for which the value of the `User-Agent` HTTP header was not associated with a standard browser. Most requests in this category come from custom applications or scripts.

Requests

The number of requests that CloudFront received from each type of operating system.

RequestsPct

The number of requests that CloudFront received from each type of operating system as a percentage of the total number of requests that CloudFront received during the time period.

Operating system trends report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

TimeBucket

The hour or the day that the data applies to, in Coordinated Universal Time (UTC).

(Operating systems)

The remaining columns in the report list the operating systems or the operating systems and their versions, depending on the value of `Grouping`. In addition to operating system names, possible values include the following:

- **Bot/Crawler** – primarily requests from search engines that are indexing your content.

- **Empty** – requests for which the value of the `User-Agent` HTTP header was empty.
- **Other** – operating systems that CloudFront identified but that aren't among the most popular. If `Bot/Crawler`, `Empty`, and/or `Unknown` don't appear among the first nine values, then they're also included in `Other`.
- **Unknown** – requests for which the operating system isn't specified in the `User-Agent` HTTP header.

Locations report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

LocationCode

The abbreviation for the location that CloudFront received requests from. For more information about possible values, see the description of Location in How data in the locations report is related to data in the CloudFront standard logs (access logs).

LocationName

The name of the location that CloudFront received requests from.

Requests

The number of requests that CloudFront received from each location.

RequestsPct

The number of requests that CloudFront received from each location as a percentage of the total number of requests that CloudFront received from all locations during the time period.

TotalBytes

The number of bytes that CloudFront served to viewers in this country or state, for the specified distribution and period.

Location trends report

The report includes the following values:

DistributionID

The ID of the distribution that you ran the report for, or `ALL` if you ran the report for all distributions.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

TimeBucket

The hour or the day that the data applies to, in Coordinated Universal Time (UTC).

(Locations)

The remaining columns in the report list the locations that CloudFront received requests from. For more information about possible values, see the description of Location in How data in the locations report is related to data in the CloudFront standard logs (access logs).

How data in the locations report is related to data in the CloudFront standard logs (access logs)

The following list shows how data in the Locations report in the CloudFront console corresponds with values in CloudFront access logs. For more information about CloudFront access logs, see Access logs (standard logs).

Location

The country or U.S. state that the viewer is in. In access logs, the `c-ip` column contains the IP address of the device that the viewer is running on. We use geolocation data to identify the geographic location of the device based on the IP address.

If you're displaying the **Locations** report by country, the country list is based on ISO 3166-2, *Codes for the representation of names of countries and their subdivisions – Part 2: Country subdivision code*. The country list includes the following additional values:

- **Anonymous Proxy** – The request originated from an anonymous proxy.

- **Satellite Provider** – The request originated from a satellite provider that provides internet service to multiple countries. Viewers might be in countries with a high risk of fraud.
- **Europe (Unknown)** – The request originated from an IP in a block that is used by multiple European countries. The country that the request originated from can't be determined. CloudFront uses **Europe (Unknown)** as the default.
- **Asia/Pacific (Unknown)** – The request originated from an IP in a block that is used by multiple countries in the Asia/Pacific region. The country that the request originated from can't be determined. CloudFront uses **Asia/Pacific (Unknown)** as the default.

If you display the **Locations** report by U.S. state, note that the report can include U.S. territories and U.S. Armed Forces regions.

> **Note**
>
> If CloudFront can't determine a user's location, the location will appear as Unknown in viewer reports.

Request Count

The total number of requests from the country or U.S. state that the viewer is in, for the specified distribution and period. This value generally corresponds closely with the number of GET requests from IP addresses in that country or state in CloudFront access logs.

Request %

One of the following, depending on the value that you selected for **Details**:

- **Countries** – The requests from this country as a percentage of the total number of requests.
- **U.S. States** – The requests from this state as a percentage of the total number of requests from the United States.

If requests came from more than 50 countries, then you can't calculate **Request %** based on the data in this table because the **Request Count** column doesn't include all of the requests during the specified period.

Bytes

The number of bytes that CloudFront served to viewers in this country or state, for the specified distribution and period. To change the display of data in this column to KB, MB, or GB, choose the link in the column heading.

Monitor CloudFront metrics with Amazon CloudWatch

Amazon CloudFront is integrated with Amazon CloudWatch and automatically publishes operational metrics for distributions and edge functions (both Lambda@Edge and CloudFront Functions). You can use these metrics to troubleshoot, track, and debug issues. Many of these metrics are displayed in a set of graphs in the CloudFront console, and are also accessible by using the CloudFront API or CLI. All of these metrics are available in the CloudWatch console or through the CloudWatch API or CLI. CloudFront metrics don't count against CloudWatch quotas (formerly known as limits) and don't incur any additional cost.

In addition to the default metrics for CloudFront distributions, you can turn on additional metrics for an additional cost. The additional metrics apply to CloudFront distributions, and must be turned on for each distribution separately. For more information about the cost, see the section called "Estimate the cost for the additional CloudFront metrics".

You can also set alarms based on these metrics in the CloudFront console, or in the CloudWatch console, API, or CLI. For example, you can set an alarm based on the `5xxErrorRate` metric, which represents the percentage of all viewer requests for which the response's HTTP status code is in the range of `500` to `599`, inclusive. When the error rate reaches a certain value for a certain amount of time, for example, 5% of requests for 5 continuous minutes, the alarm is triggered. You specify the alarm's value and its time unit when you create the alarm.

Notes

- When you create a CloudWatch alarm in the CloudFront console, it creates one for you in the US East (N. Virginia) Region (`us-east-1`). If you create an alarm from the CloudWatch console, you must use the same Region. Because CloudFront is a global service, metrics for the service are sent to US East (N. Virginia).
- When creating alarms, standard CloudWatch pricing applies.

Topics

- View CloudFront and edge function metrics
- Create alarms for metrics
- Download metrics data in CSV format

- Types of metrics for CloudFront

View CloudFront and edge function metrics

You can view operational metrics about your CloudFront distributions and edge functions in the CloudFront console.

To view CloudFront and edge function metrics in CloudFront

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Monitoring**.
3. To view graphs about the activity for a specific CloudFront distribution or edge function, choose one, and then choose **View distribution metrics** or **View metrics**.
4. You can customize the graphs by doing the following:
   a. To change the time range for the information displayed in the graphs, choose 1h (1 hour), 3h (3 hours), or another range, or specify a custom range.
   b. To change how often CloudFront updates the information in the graph, choose the down arrow next to the refresh icon, and then choose a refresh rate. The default refresh rate is 1 minute, but you can choose other options.
5. To view CloudFront graphs in the CloudWatch console, choose **Add to dashboard**. You must use the US East (N. Virginia) Region to view the graphs in the CloudWatch console.

Topics

- Default CloudFront distribution metrics
- Turn on additional CloudFront distribution metrics
- Default Lambda@Edge function metrics
- Default CloudFront Functions metrics

Default CloudFront distribution metrics

The following default metrics are included for all CloudFront distributions, at no additional cost:

Requests

The total number of viewer requests received by CloudFront, for all HTTP methods and for both HTTP and HTTPS requests.

Bytes downloaded

The total number of bytes downloaded by viewers for `GET` and `HEAD` requests.

Bytes uploaded

The total number of bytes that viewers uploaded to CloudFront, using `OPTIONS`, `POST` and `PUT` requests.

4xx error rate

The percentage of all viewer requests for which the response's HTTP status code is `4xx`.

5xx error rate

The percentage of all viewer requests for which the response's HTTP status code is `5xx`.

Total error rate

The percentage of all viewer requests for which the response's HTTP status code is `4xx` or `5xx`.

These metrics are shown in graphs for each CloudFront distribution on the **Monitoring** page of the CloudFront console. On each graph, the totals are displayed at 1-minute granularity. In addition to viewing the graphs, you can also download metrics reports as CSV files.

Turn on additional CloudFront distribution metrics

In addition to the default metrics, you can turn on additional metrics for an additional cost. For more information about the cost, see the section called "Estimate the cost for the additional CloudFront metrics".

These additional metrics must be turned on for each distribution separately:

Cache hit rate

The percentage of all cacheable requests for which CloudFront served the content from its cache. HTTP `POST` and `PUT` requests, and errors, are not considered cacheable requests.

Origin latency

The total time spent from when CloudFront receives a request to when it starts providing a response to the network (not the viewer), for requests that are served from the origin, not the CloudFront cache. This is also known as *first byte latency*, or *time-to-first-byte*.

Error rate by status code

The percentage of all viewer requests for which the response's HTTP status code is a particular code in the `4xx` or `5xx` range. This metric is available for all of the following error codes: `401`, `403`, `404`, `502`, `503`, and `504`.

You can turn on additional metrics in the CloudFront console, with CloudFormation, with the AWS Command Line Interface (AWS CLI), or with the CloudFront API.

Console

To turn on additional metrics

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Monitoring**.
3. Choose the distribution to turn on additional metrics for, and then choose **View distribution metrics**.
4. Choose **Manage additional metrics**.
5. In the **Manage additional metrics** window, turn on **Enabled**. After you turn on the additional metrics, you can close the **Manage additional metrics** window.

   After you turn on the additional metrics, they are shown in graphs. On each graph, the totals are displayed at 1-minute granularity. In addition to viewing the graphs, you can also download metrics reports as CSV files.

CloudFormation

To turn on additional metrics with CloudFormation, use the `AWS::CloudFront::MonitoringSubscription` resource type. The following example shows the CloudFormation template syntax, in YAML format, for enabling additional metrics.

```
Type: AWS::CloudFront::MonitoringSubscription
```

```
Properties:
  DistributionId: EDFDVBD6EXAMPLE
  MonitoringSubscription:
    RealtimeMetricsSubscriptionConfig:
      RealtimeMetricsSubscriptionStatus: Enabled
```

CLI

To manage additional metrics with the AWS Command Line Interface (AWS CLI), use one of the following commands:

To turn on additional metrics for a distribution

- Use the **create-monitoring-subscription** command, as in the following example. Replace *EDFDVBD6EXAMPLE* with the ID of the distribution that you're enabling additional metrics for.

```
aws cloudfront create-monitoring-subscription --
distribution-id EDFDVBD6EXAMPLE --monitoring-subscription
 RealtimeMetricsSubscriptionConfig={RealtimeMetricsSubscriptionStatus=Enabled}
```

To see whether additional metrics are turned on for a distribution

- Use the **get-monitoring-subscription** command, as in the following example. Replace *EDFDVBD6EXAMPLE* with the ID of the distribution that you're checking.

```
aws cloudfront get-monitoring-subscription --distribution-id EDFDVBD6EXAMPLE
```

To turn off additional metrics for a distribution

- Use the **delete-monitoring-subscription** command, as in the following example. Replace *EDFDVBD6EXAMPLE* with the ID of the distribution that you're turning off additional metrics for.

```
aws cloudfront delete-monitoring-subscription --distribution-id EDFDVBD6EXAMPLE
```

API

To manage additional metrics with the CloudFront API, use one of the following API operations.

- To turn on additional metrics for a distribution, use CreateMonitoringSubscription.
- To see whether additional metrics are turned on for a distribution, use GetMonitoringSubscription.
- To turn off additional metrics for a distribution, use DeleteMonitoringSubscription.

For more information about these API operations, see the API reference documentation for your AWS SDK or other API client.

Estimate the cost for the additional CloudFront metrics

When you turn on additional metrics for a distribution, CloudFront sends up to 8 metrics to CloudWatch in the US East (N. Virginia) Region. CloudWatch charges a low, fixed rate for each metric. This rate is charged only once per month, per metric (up to 8 metrics per distribution). This is a fixed rate, so your cost remains the same regardless of the number of requests or responses that the CloudFront distribution receives or sends. For the per-metric rate, see the Amazon CloudWatch pricing page and the CloudWatch pricing calculator. Additional API charges apply when you retrieve the metrics with the CloudWatch API.

Default Lambda@Edge function metrics

You can use CloudWatch metrics to monitor, in real time, problems with your Lambda@Edge functions. There's no additional charge for these metrics.

When you attach a Lambda@Edge function to a cache behavior in a CloudFront distribution, Lambda begins sending metrics to CloudWatch automatically. Metrics are available for all Lambda Regions, but to view metrics in the CloudWatch console or get the metric data from the CloudWatch API, you must use the US East (N. Virginia) Region (`us-east-1`). The metric group name is formatted as: `AWS/CloudFront/`*`distribution-ID`*, where *`distribution-ID`* is the ID of the CloudFront distribution that the Lambda@Edge function is associated with. For more information about CloudWatch metrics, see the Amazon CloudWatch User Guide.

The following default metrics are shown in graphs for each Lambda@Edge function on the **Monitoring** page of the CloudFront console:

- 5xx error rate for Lambda@Edge
- Lambda execution errors
- Lambda invalid responses
- Lambda throttles

The graphs include the number of invocations, errors, throttles, and so on. On each graph, the totals are displayed at 1-minute granularity, grouped by AWS Region.

If you see a spike in errors that you want to investigate, you can choose a function and then view log files by AWS Region, until you determine which function is causing the problems and in which AWS Region. For more information about troubleshooting Lambda@Edge errors, see:

- the section called "How to determine the type of failure"
- Four Steps for Debugging your Content Delivery on AWS

Default CloudFront Functions metrics

CloudFront Functions sends operational metrics to Amazon CloudWatch so that you can monitor your functions. Viewing these metrics can help you troubleshoot, track, and debug issues. CloudFront Functions publishes the following metrics to CloudWatch:

- **Invocations** (`FunctionInvocations`) – The number of times the function was started (invoked) in a given time period.
- **Validation errors** (`FunctionValidationErrors`) – The number of validation errors produced by the function in a given time period. Validation errors occur when the function runs successfully but returns invalid data (an invalid event object).
- **Execution errors** (`FunctionExecutionErrors`) – The number of execution errors that occurred in a given time period. Execution errors occur when the function fails to complete successfully.
- **Compute utilization** (`FunctionComputeUtilization`) – The amount of time that the function took to run as a percentage of the maximum allowed time. For example, a value of 35 means that the function completed in 35% of the maximum allowed time. This metric is a number between 0 and 100.

  If this value reaches or is near 100, the function has used or is close to using the allowed execution time and subsequent requests might be throttled. If your function is running at 80% or more utilization, we recommend that you review your function to reduce the execution time and

improve utilization. For example, you might want to only log errors, simplify any complex regex expressions, or remove unnecessary parsing of complex JSON objects.

- **Throttles** (`FunctionThrottles`) – The number of times that the function was throttled in a given time period. Functions can be throttled for the following reasons:
  - The function continuously exceeds the maximum time allowed for execution
  - The function results in compilation errors
  - There is an unusually high number of requests per second

CloudFront KeyValueStore also sends the following operational metrics to Amazon CloudWatch:

- **Read requests** (`KvsReadRequests`) – The number of times the function successfully read from the key value store within a given time period.
- **Read errors** (`KvsReadErrors`) – The number of times the function failed to read from the key value store within a given time period.

All of these metrics are published to CloudWatch in the US East (N. Virginia) Region (`us-east-1`), in the CloudFront namespace. You can also view these metrics in the CloudWatch console. In the CloudWatch console, you can view the metrics per function or per function per distribution.

You can also use CloudWatch to set alarms based on these metrics. For example, you can set an alarm based on the execution time (`FunctionComputeUtilization`) metric, which represents the percentage of available time that your function took to run. When the execution time reaches a certain value for a certain amount of time. For example, if you choose greater than 70% of available time for 15 continuous minutes, the alarm is triggered. You specify the alarms value and its time unit when you create the alarm.

> **Note**
>
> CloudFront Functions sends metrics to CloudWatch only for functions in the `LIVE` stage that run in response to production requests and responses. When you test a function, CloudFront doesn't send any metrics to CloudWatch. The test output contains information about errors, compute utilization, and function logs (`console.log()` statements), but this information isn't sent to CloudWatch.

For information about how to get these metrics with the CloudWatch API, see the section called "CloudFront metrics".

Create alarms for metrics

In the CloudFront console, you can set alarms to notify you by Amazon Simple Notification Service (Amazon SNS) based on specific CloudFront metrics.

To create alarms for metrics

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Alarms**.
3. Choose **Create alarm**.
4. For **Details**, specify the following:
   a. **Alarm name** – A name for the alarm.
   b. **Distribution** – The CloudFront distribution that you're creating the alarm for.
5. For **Condition**, specify the following:
   a. **Metric** – The metric that you're creating the alarm for.
   b. **"IF" <condition>** – The threshold when CloudWatch should trigger an alarm and send a notification to the Amazon SNS topic. For example, to receive a notification when the 5xx error rate exceeds 1%, specify the following:

      5xx error rate > 1

   c. **"FOR" consecutive periods** – The period of time that the condition must be met before triggering an alarm. When you choose a value, aim for an appropriate balance between a value that doesn't alarm for temporary problems, but will alarm for sustained or real problems.
   d. (Optional) **Notify** – The Amazon SNS topic to send notification to if this metric triggers an alarm.
6. Choose **Create alarm**.

Notes

- When you enter the values for the condition, use whole numbers without punctuation. For example, to specify one thousand, enter **1000**.
- For 4xx, 5xx, and total error rates, the value that you specify is a percentage.
- For requests, bytes downloaded, and bytes uploaded, the value that you specify is units. For example, 1073742000 bytes.

For more information about creating Amazon SNS topics, see Creating an Amazon SNS topic in the *Amazon Simple Notification Service Developer Guide*.

Download metrics data in CSV format

You can download the CloudWatch metrics data for a CloudFront distribution in CSV format.

To download metrics data in CSV format

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. In the navigation pane, choose **Monitoring**.
3. Choose the distribution and then choose **View distribution metrics**.
4. Choose **Download CSV** and then choose the time period (for example, **For last 1 day (1 hour period)**.
5. After your file downloads, open it to view the following information.

Topics

- Information about the report
- Data in the metrics report

Information about the report

The first few rows of the report include the following information:

Version

The CloudFront reporting version.

Report

The name of the report.

DistributionID

The ID of the distribution for which you ran the report.

StartDateUTC

The beginning of the date range for which you ran the report, in Coordinated Universal Time (UTC).

EndDateUTC

The end of the date range for which you ran the report, in Coordinated Universal Time (UTC).

GeneratedTimeUTC

The date and time on which you ran the report, in Coordinated Universal Time (UTC).

Granularity

The time period for each row in the report, for example, `ONE_MINUTE`.

Data in the metrics report

The report includes the following values:

DistributionID

The ID of the distribution for which you ran the report.

FriendlyName

An alternate domain name (CNAME) for the distribution, if any. If a distribution has no alternate domain names, the list includes an origin domain name for the distribution.

TimeBucket

The hour or the day that the data applies to, in Coordinated Universal Time (UTC).

Requests

The total number of requests for all HTTP status codes (for example, `200`, `404`, and so on) and all methods (for example, `GET`, `HEAD`, `POST`, and so on) during the time period.

BytesDownloaded

The number of bytes that viewers downloaded for the specified distribution during the time period.

BytesUploaded

The number of bytes that viewers uploaded for the specified distribution during the time period.

TotalErrorRatePct

The percentage of requests for which the HTTP status code was a `4xx` or `5xx` error for the specified distribution during the time period.

4xxErrorRatePct

The percentage of requests for which the HTTP status code was a `4xx` error for the specified distribution during the time period.

5xxErrorRatePct

The percentage of requests for which the HTTP status code was a `5xx` error for the specified distribution during the time period.

If you have turned on additional metrics for your distribution, then the report also includes the following additional values:

401ErrorRatePct

The percentage of requests for which the HTTP status code was a `401` error for the specified distribution during the time period.

403ErrorRatePct

The percentage of requests for which the HTTP status code was a `403` error for the specified distribution during the time period.

404ErrorRatePct

The percentage of requests for which the HTTP status code was a 404 error for the specified distribution during the time period.

502ErrorRatePct

The percentage of requests for which the HTTP status code was a 502 error for the specified distribution during the time period.

503ErrorRatePct

The percentage of requests for which the HTTP status code was a 503 error for the specified distribution during the time period.

504ErrorRatePct

The percentage of requests for which the HTTP status code was a 504 error for the specified distribution during the time period.

OriginLatency

The total time spent, in milliseconds, from when CloudFront received a request to when it started providing a response to the network (not the viewer), for requests that were served from the origin, not the CloudFront cache. This is also known as *first byte latency*, or *time-to-first-byte*.

CacheHitRate

The percentage of all cacheable requests for which CloudFront served the content from its cache. HTTP POST and PUT requests, and errors, are not considered cacheable requests.

Types of metrics for CloudFront

You can use the CloudWatch API or AWS Command Line Interface (AWS CLI) to get the CloudFront metrics in programs or applications that you build. You can use the raw data to build your own custom dashboards, your own alarming tools, and so on.

For more information, see get-metric-data in the *AWS CLI Command Reference* or the GetMetricData API operation in the *Amazon CloudWatch API Reference*.

Topics

- Values for all CloudFront metrics

- Values for CloudFront distribution metrics
- Values for CloudFront function metrics

> **Note**
>
> To get the CloudFront metrics from the CloudWatch API, you must use the US East (N. Virginia) Region (`us-east-1`). You also need to know certain values and types for each metric.

Values for all CloudFront metrics

The following values apply to all CloudFront metrics:

`Namespace`

The value for `Namespace` is always `AWS/CloudFront`.

Dimensions

Each CloudFront metric has the following dimensions:

`DistributionId`

The ID of the CloudFront distribution for which you want to get metrics.

`FunctionName`

The name of the function (in CloudFront Functions) for which you want to get metrics.

This dimension applies only to functions.

`Region`

The value for `Region` is always `Global`, because CloudFront is a global service.

Values for CloudFront distribution metrics

Use information from the following list to get details about specific CloudFront distribution metrics from the CloudWatch API. Some of these metrics are available only when you have turned on additional metrics for the distribution.

> **Note**
>
> Only one statistic, `Average` or `Sum`, is applicable for each metric. The following list specifies which statistic is applicable to that metric.

4xx error rate

The percentage of all viewer requests for which the response's HTTP status code is `4xx`.

- Metric name: `4xxErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

401 error rate

The percentage of all viewer requests for which the response's HTTP status code is `401`. To get this metric, you must first turn on additional metrics.

- Metric name: `401ErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

403 error rate

The percentage of all viewer requests for which the response's HTTP status code is `403`. To get this metric, you must first turn on additional metrics.

- Metric name: `403ErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

404 error rate

The percentage of all viewer requests for which the response's HTTP status code is `404`. To get this metric, you must first turn on additional metrics.

- Metric name: `404ErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

5xx error rate

The percentage of all viewer requests for which the response's HTTP status code is `5xx`.

- Metric name: `5xxErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

502 error rate

The percentage of all viewer requests for which the response's HTTP status code is `502`. To get this metric, you must first turn on additional metrics.

- Metric name: `502ErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

503 error rate

The percentage of all viewer requests for which the response's HTTP status code is `503`. To get this metric, you must first turn on additional metrics.

- Metric name: `503ErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

504 error rate

The percentage of all viewer requests for which the response's HTTP status code is `504`. To get this metric, you must first turn on additional metrics.

- Metric name: `504ErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

Bytes downloaded

The total number of bytes downloaded by viewers for `GET` and `HEAD` requests.

- Metric name: `BytesDownloaded`
- Valid statistic: `Sum`
- Unit: `None`

Bytes uploaded

The total number of bytes that viewers uploaded to CloudFront, using `OPTIONS`, `POST` and `PUT` requests.

- Metric name: `BytesUploaded`
- Valid statistic: `Sum`
- Unit: None

Cache hit rate

The percentage of all cacheable requests for which CloudFront served the content from its cache. HTTP POST and PUT requests, and errors, are not considered cacheable requests. To get this metric, you must first turn on additional metrics.

- Metric name: `CacheHitRate`
- Valid statistic: `Average`
- Unit: `Percent`

Origin latency

The total time spent, in milliseconds, from when CloudFront receives a request to when it starts providing a response to the network (not the viewer), for requests that are served from the origin, not the CloudFront cache. This is also known as *first byte latency*, or *time-to-first-byte*. To get this metric, you must first turn on additional metrics.

- Metric name: `OriginLatency`
- Valid statistic: `Percentile`
- Unit: `Milliseconds`

> **Note**
>
> To get a `Percentile` statistic from the CloudWatch API, use the `ExtendedStatistics` parameter, not `Statistics`. For more information, see GetMetricStatistics in the *Amazon CloudWatch API Reference*, or the reference documentation for the AWS SDKs.

Requests

The total number of viewer requests received by CloudFront, for all HTTP methods and for both HTTP and HTTPS requests.

- Metric name: `Requests`
- Valid statistic: `Sum`
- Unit: None

Total error rate

The percentage of all viewer requests for which the response's HTTP status code is `4xx` or `5xx`.

- Metric name: `TotalErrorRate`
- Valid statistic: `Average`
- Unit: `Percent`

Values for CloudFront function metrics

Use information from the following list to get details about specific CloudFront function metrics from the CloudWatch API.

> **Note**
>
> Only one statistic, `Average` or `Sum`, is applicable for each metric. The following list specifies which statistic is applicable to that metric.

Invocations

The number of times the function was started (invoked) in a given time period.

- Metric name: `FunctionInvocations`
- Valid statistic: `Sum`
- Unit: `None`

Validation errors

The number of validation errors produced by the function in a given time period. Validation errors occur when the function runs successfully but returns invalid data (an invalid event object).

- Metric name: `FunctionValidationErrors`
- Valid statistic: `Sum`
- Unit: `None`

Execution errors

The number of execution errors that occurred in a given time period. Execution errors occur when the function fails to complete successfully.

- Metric name: `FunctionExecutionErrors`
- Valid statistic: `Sum`
- Unit: None

Compute utilization

The amount of time (0-100) that the function took to run as a percentage of the maximum allowed time. For example, a value of 35 means that the function completed in 35% of the maximum allowed time.

- Metric name: `FunctionComputeUtilization`
- Valid statistic: `Average`
- Unit: `Percent`

Throttles

The number of times that the function was throttled in a given time period.

- Metric name: `FunctionThrottles`
- Valid statistic: `Sum`
- Unit: None

CloudFront and edge function logging

Amazon CloudFront provides different kinds of logging. You can log the viewer requests that come to your CloudFront distributions, or you can log the CloudFront service activity (API activity) in your AWS account. You can also get logs from your CloudFront Functions and Lambda@Edge functions.

Logging requests

CloudFront provides the following ways to log the requests that come to your distributions.

Access logs (standard logs)

CloudFront access logs provide detailed records about every request that's made to a distribution. You can use the logs for scenarios, such as security and access audits.

CloudFront access logs are delivered to the delivery destination that you specify.

Use access logs when you need:

- Historical analysis and reporting
- Security audits and compliance requirements
- Cost-effective long-term log retention

For more information, see Access logs (standard logs).

Real-time access logs

CloudFront real-time access logs are delivered within seconds of receiving the requests and provide information about requests made to a distribution in real time. You can choose the *sampling rate* for your real-time access logs—that is, the percentage of requests for which you want to receive real-time access log records. You can also choose the specific fields that you want to receive in the log records. Real-time access logs are ideal for live monitoring for content delivery performance.

CloudFront real-time access logs are delivered to the data stream of your choice in Amazon Kinesis Data Streams. CloudFront charges for real-time access logs, in addition to the charges you incur for using Kinesis Data Streams.

Use real-time access logs when you need:

- Real-time monitoring and alerts
- Live dashboards and operational insights

For more information, see Use real-time access logs.

Connection logs

Connection logs provide detailed information about the TLS connection between the server and the client. CloudFront generates them only for connections to mTLS enabled distributions, and for HTTPS connections to distribution tenants on the Anycast static IP addresses of their connection group. For mTLS enabled distributions, Connection logs also provide visibility into client certificate information, reasons for mTLS authentication failures and whether a connection was permitted or refused.

Like access logs (standard logs), connection logs are delivered to the delivery destination that you specify.

Use connection logs when you need:

- Reasons for successful or unsuccessful connections during the TLS handshake
- Visibility into the client certificate information

For more information, see Connection logs.

Logging edge functions

You can use Amazon CloudWatch Logs to get logs for your edge functions, both Lambda@Edge and CloudFront Functions. You can access the logs using the CloudWatch console or the CloudWatch Logs API. For more information, see the section called “Edge function logs”.

Logging service activity

You can use AWS CloudTrail to log the CloudFront service activity (API activity) in your AWS account. CloudTrail provides a record of API actions taken by a user, role, or AWS service in CloudFront. Using the information collected by CloudTrail, you can determine the API request that was made to CloudFront, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information, see Logging Amazon CloudFront API calls using AWS CloudTrail.

For more information about logging, see the following topics:

Topics

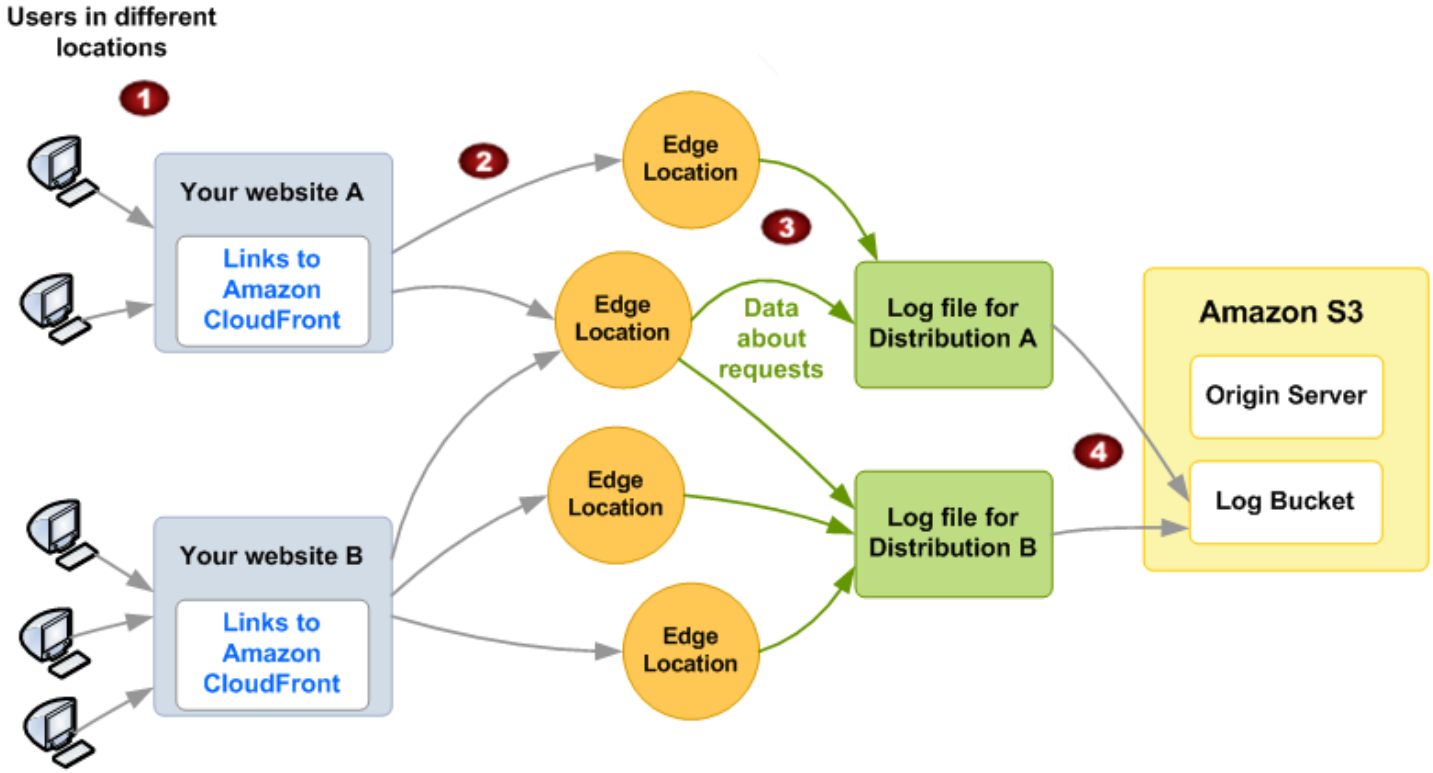
- Access logs (standard logs)
- Use real-time access logs
- Connection logs
- Edge function logs
- Logging Amazon CloudFront API calls using AWS CloudTrail

Access logs (standard logs)

You can configure CloudFront to create log files that contain detailed information about every user (viewer) request that CloudFront receives. These are called *access logs*, also known as *standard logs*.

Each log contains information such as the time the request was received, the processing time, request paths, and server responses. You can use these access logs to analyze response times and to troubleshoot issues.

The following diagram shows how CloudFront logs information about requests for your objects. In this example, the distributions are configured to send access logs to an Amazon S3 bucket.



1. In this example, you have two websites, A and B, and two corresponding CloudFront distributions. Users request your objects using URLs that are associated with your distributions.
2. CloudFront routes each request to the appropriate edge location.
3. CloudFront writes data about each request to a log file specific to that distribution. In this example, information about requests related to Distribution A goes into a log file for Distribution A. Information about requests related to Distribution B goes into a log file for Distribution B.
4. CloudFront periodically saves the log file for a distribution in the Amazon S3 bucket that you specified when you enabled logging. CloudFront then starts saving information about subsequent requests in a new log file for the distribution.

   If viewers don't access your content during a given hour, you don't receive any log files for that hour.

> **Note**
>
> We recommend that you use the logs to understand the nature of the requests for your content, not as a complete accounting of all requests. CloudFront delivers access logs on a best-effort basis. The log entry for a particular request might be delivered long after

the request was actually processed. In rare cases, a log entry might not be delivered at all. When a log entry is omitted from access logs, the number of entries in the access logs won't match the usage that appears in the AWS billing and usage reports.

CloudFront supports two versions of standard logging. Standard logging (legacy) supports sending your access logs to Amazon S3 *only*. Standard logging (v2) supports additional delivery destinations. You can configure both or either logging option for your distribution. For more information, see the following topics:

Topics

- Configure standard logging (v2)
- Configure standard logging (legacy)
- Standard logging reference

Tip

CloudFront also offers real-time access logs, which give you information about requests made to a distribution in real time (logs are delivered within seconds of receiving the requests). You can use real-time access logs to monitor, analyze, and take action based on content delivery performance. For more information, see Use real-time access logs.

Configure standard logging (v2)

You can enable access logs (standard logs) when you create or update a distribution. Standard logging (v2) includes the following features:

- Send access logs to Amazon CloudWatch Logs, Amazon Data Firehose, and Amazon Simple Storage Service (Amazon S3).
- Select the log fields that you want. You can also select a subset of real-time access log fields.
- Select additional output log file formats.

If you're using Amazon S3, you have the following optional features:

- Send logs to opt-in AWS Regions.

- Organize your logs with partitioning.
- Enable Hive-compatible file names.

For more information, see Send logs to Amazon S3.

To get started with standard logging, complete the following steps:

1. Set up your required permissions for the specified AWS service that will receive your logs.
2. Configure standard logging from the CloudFront console or the CloudWatch API.
3. View your access logs.

> **Note**
>
> - If you enable standard logging (v2), this doesn't affect or change standard logging (legacy). You can continue to use standard logging (legacy) for your distribution, in addition to using standard logging (v2). For more information, see Configure standard logging (legacy).
> - If you already enabled standard logging (legacy) and you want to enable standard logging (v2) to Amazon S3, we recommend that you specify a *different* Amazon S3 bucket or use a *separate path* in the same bucket (for example, use a log prefix or partitioning). This helps you keep track of which log files are associated with which distribution and prevents log files from overwriting each other.

Permissions

CloudFront uses CloudWatch vended logs to deliver access logs. To do so, you need permissions to the specified AWS service so that you can enable logging delivery.

To see the required permissions for each logging destination, choose from the following topics in the *Amazon CloudWatch Logs User Guide*.

- CloudWatch Logs
- Firehose
- Amazon S3

After you have set up permissions to your logging destination, you can enable standard logging for your distribution.

> **Note**
>
> CloudFront supports sending access logs to different AWS accounts (cross accounts). To enable cross-account delivery, both accounts (your account and the receiving account) must have the required permissions. For more information, see the Enable standard logging for cross-account delivery section or the Cross-account delivery example in the *Amazon CloudWatch Logs User Guide*.

Enable standard logging

To enable standard logging, you can use the CloudFront console or the CloudWatch API.

Contents

- Enable standard logging (CloudFront console)
- Enable standard logging (CloudWatch API)

Enable standard logging (CloudFront console)

To enable standard logging for a CloudFront distribution (console)

1. Use the CloudFront console to update an existing distribution.
2. Choose the **Logging** tab.
3. Choose **Add**, then select the service to receive your logs:
   - CloudWatch Logs
   - Firehose
   - Amazon S3
4. For the **Destination**, select the resource for your service. If you haven't already created your resource, you can choose **Create** or see the following documentation.
   - For CloudWatch Logs, enter the **Log group name**.
   - For Firehose, enter the **Firehose delivery stream**.
   - For Amazon S3, enter the **Bucket name**.

> **Tip**
>
> To specify a prefix, enter the prefix after the bucket name, such as `amzn-s3-demo-bucket.s3.amazonaws.com/MyLogPrefix`. If you don't specify a prefix, CloudFront will automatically add one for you. For more information, see Send logs to Amazon S3.

5. For **Additional settings – *optional***, you can specify the following options:

   a. For **Field selection**, select the log field names that you want to deliver to your destination. You can select access log fields and a subset of real-time access log fields.

   b. (Amazon S3 only) For **Partitioning**, specify the path to partition your log file data.

   c. (Amazon S3 only) For **Hive-compatible file format**, you can select the checkbox to use Hive-compatible S3 paths. This helps simplify loading new data into your Hive-compatible tools.

   d. For **Output format**, specify your preferred format.

   > **Note**
   >
   > If you choose **Parquet**, this option incurs CloudWatch charges for converting your access logs to Apache Parquet. For more information, see the Vended Logs section for CloudWatch pricing.

   e. For **Field delimiter**, specify how to separate log fields.

6. Complete the steps to update or create your distribution.
7. To add another destination, repeat steps 3–6.
8. From the **Logs** page, verify that the standard logs status is **Enabled** next to the distribution.
9. (Optional) To enable cookie logging, choose **Manage**, **Settings** and turn on **Cookie logging**, then choose **Save changes**.

> **Tip**
>
> Cookie logging is a global setting that applies to *all* standard logging for your distribution. You can't override this setting for separate delivery destinations.

For more information about the standard logging delivery and log fields, see the Standard logging reference.

Enable standard logging (CloudWatch API)

You can also use the CloudWatch API to enable standard logging for your distributions.

Notes

- When calling the CloudWatch API to enable standard logging, you must specify the US East (N. Virginia) Region (`us-east-1`), even if you want to enable cross Region delivery to another destination. For example, if you want to send your access logs to an S3 bucket in the Europe (Ireland) Region (`eu-west-1`), use the CloudWatch API in the `us-east-1` Region.
- There is an additional option to include cookies in standard logging. In the CloudFront API, this is the `IncludeCookies` parameter. If you configure access logging by using the CloudWatch API and you specify that you want to include cookies, you must use the CloudFront console or CloudFront API to update your distribution to include cookies. Otherwise, CloudFront can't send cookies to your log destination. For more information, see Cookie logging.

To enable standard logging for a distribution (CloudWatch API)

1. After you create a distribution, get the Amazon Resource Name (ARN).

   You can find the ARN from the **Distribution** page in the CloudFront console or you can use the GetDistribution API operation. A distribution ARN follows the format: `arn:aws:cloudfront::123456789012:distribution/d111111abcdef8`
2. Next, use the CloudWatch PutDeliverySource API operation to create a delivery source for the distribution.

   a. Enter a name for the delivery source.

   b. Pass the `resourceArn` of the distribution.

   c. For `logType`, specify `ACCESS_LOGS` as the type of logs that are collected.

   d. **Example AWS CLI put-delivery-source command**

   The following is an example of configuring a delivery source for a distribution.

```
aws logs put-delivery-source --name S3-delivery --resource-arn
 arn:aws:cloudfront::123456789012:distribution/d111111abcdef8 --log-type
 ACCESS_LOGS
```

Output

```
{
 "deliverySource": {
 "name": "S3-delivery",
 "arn": "arn:aws:logs:us-east-1:123456789012:delivery-source:S3-delivery",
 "resourceArns": [
 "arn:aws:cloudfront::123456789012:distribution/d111111abcdef8"
 ],
 "service": "cloudfront",
 "logType": "ACCESS_LOGS"
 }
}
```

3. Use the PutDeliveryDestination API operation to configure where to store your logs.

   a. For `destinationResourceArn`, specify the ARN of the destination. This can be a CloudWatch Logs log group, a Firehose delivery stream, or an Amazon S3 bucket.

   b. For `outputFormat`, specify the output format for your logs.

   c. **Example AWS CLI put-delivery-destination command**

   The following is an example of configuring a delivery destination to an Amazon S3 bucket.

```
aws logs put-delivery-destination --name S3-destination --delivery-destination-
configuration destinationResourceArn=arn:aws:s3:::amzn-s3-demo-bucket
```

Output

```
{
    "name": "S3-destination",
    "arn": "arn:aws:logs:us-east-1:123456789012:delivery-destination:S3-
destination",
    "deliveryDestinationType": "S3",
    "deliveryDestinationConfiguration": {
        "destinationResourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
    }
```

```
}
```

> **Note**
>
> If you're delivering logs cross-account, you must use the PutDeliveryDestinationPolicy API operation to assign an AWS Identity and Access Management (IAM) policy to the destination account. The IAM policy allows delivery from one account to another account.

4. Use the CreateDelivery API operation to link the delivery source to the destination that you created in the previous steps. This API operation associates the delivery source with the end destination.

   a. For `deliverySourceName`, specify the source name.

   b. For `deliveryDestinationArn`, specify the ARN for the delivery destination.

   c. For `fieldDelimiter`, specify the string to separate each log field.

   d. For `recordFields`, specify the log fields that you want.

   e. If you're using S3, specify whether to use `enableHiveCompatiblePath` and `suffixPath`.

Example AWS CLI create-delivery command

The following is an example of creating a delivery.

```
aws logs create-delivery --delivery-source-name cf-delivery --delivery-destination-
arn arn:aws:logs:us-east-1:123456789012:delivery-destination:S3-destination
```

Output

```
{
    "id": "abcNegnBoTR123",
    "arn": "arn:aws:logs:us-east-1:123456789012:delivery:abcNegnBoTR123",
    "deliverySourceName": "cf-delivery",
    "deliveryDestinationArn": "arn:aws:logs:us-east-1:123456789012:delivery-
destination:S3-destination",
    "deliveryDestinationType": "S3",
    "recordFields": [
```

```
        "date",
        "time",
        "x-edge-location",
        "sc-bytes",
        "c-ip",
        "cs-method",
        "cs(Host)",
        "cs-uri-stem",
        "sc-status",
        "cs(Referer)",
        "cs(User-Agent)",
        "cs-uri-query",
        "cs(Cookie)",
        "x-edge-result-type",
        "x-edge-request-id",
        "x-host-header",
        "cs-protocol",
        "cs-bytes",
        "time-taken",
        "x-forwarded-for",
        "ssl-protocol",
        "ssl-cipher",
        "x-edge-response-result-type",
        "cs-protocol-version",
        "fle-status",
        "fle-encrypted-fields",
        "c-port",
        "time-to-first-byte",
        "x-edge-detailed-result-type",
        "sc-content-type",
        "sc-content-len",
        "sc-range-start",
        "sc-range-end",
        "c-country",
        "cache-behavior-path-pattern"
    ],
     "fieldDelimiter": ""
}
```

5. From the CloudFront console, on the **Logs** page, verify that the standard logs status is **Enabled** next to the distribution.

For more information about the standard logging delivery and log fields, see the Standard logging reference.

> **Note**
>
> To enable standard logging (v2) for CloudFront by using AWS CloudFormation, you can use the following CloudWatch Logs properties:
>
> - Delivery
> - DeliveryDestination
> - DeliverySource
>
> The `ResourceArn` is the CloudFront distribution and `LogType` must be `ACCESS_LOGS` as the supported log type.

Enable standard logging for cross-account delivery

If you enable standard logging for your AWS account and you want to deliver your access logs to another account, make sure that you configure the source account and the destination account correctly. The *source account* with the CloudFront distribution sends its access logs to the *destination account*.

In this example procedure, the source account *111111111111*) sends its access logs to an Amazon S3 bucket in the destination account (*222222222222*). To send access logs to an Amazon S3 bucket in the destination account, use the AWS CLI.

Configure the destination account

For destination account, complete the following procedure.

To configure the destination account

1. To create the log delivery destination, you can enter the following AWS CLI command. This example uses the *`MyLogPrefix`* string to create a prefix for your access logs.

```
aws logs put-delivery-destination --name cloudfront-delivery-destination --
delivery-destination-configuration "destinationResourceArn=arn:aws:s3:::amzn-s3-
demo-bucket-cloudfront-logs/MyLogPrefix"
```

Output

```
{
    "deliveryDestination": {
        "name": "cloudfront-delivery-destination",
        "arn": "arn:aws:logs:us-east-1:222222222222:delivery-
destination:cloudfront-delivery-destination",
        "deliveryDestinationType": "S3",
        "deliveryDestinationConfiguration": {"destinationResourceArn":
 "arn:aws:s3:::amzn-s3-demo-bucket-cloudfront-logs/MyLogPrefix"}
    }
}
```

> **Note**
>
> If you specify an S3 bucket *without* a prefix, CloudFront will automatically append the `AWSLogs/<account-ID>/CloudFront` as a prefix that appears in the `suffixPath` of the S3 delivery destination. For more information, see S3DeliveryConfiguration.

2. Add the resource policy for the log delivery destination to allow the source account to create a log delivery.

   In the following policy, replace *111111111111* with the source account ID and specify the delivery destination ARN from the output in step 1.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateDelivery",
            "Effect": "Allow",
            "Principal": {"AWS": "111111111111"},
            "Action": ["logs:CreateDelivery"],
```

```
            "Resource": "arn:aws-eusc:logs:eusc-de-
east-1:222222222222:delivery-destination:cloudfront-delivery-destination"
        }
    ]
}
```

3. Save the file, such as `deliverypolicy.json`.
4. To attach the previous policy to the delivery destination, enter the following AWS CLI command.

```
aws logs put-delivery-destination-policy --delivery-destination-name cloudfront-
delivery-destination --delivery-destination-policy file://deliverypolicy.json
```

5. Add the below statement to the destination Amazon S3 bucket policy, replacing the resource ARN and the source account ID. This policy allows the `delivery.logs.amazonaws.com` service principal to perform the `s3:PutObject` action.

```
{
    "Sid": "AWSLogsDeliveryWrite",
    "Effect": "Allow",
    "Principal": {"Service": "delivery.logs.amazonaws.com"},
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket-cloudfront-logs/*",
    "Condition": {
        "StringEquals": {
            "s3:x-amz-acl": "bucket-owner-full-control",
            "aws:SourceAccount": "111111111111"
        },
        "ArnLike": {"aws:SourceArn": "arn:aws:logs:us-east-1:111111111111:delivery-
source:*"}
    }
}
```

6. If you're using AWS KMS for your bucket, add the following statement to the KMS key policy to grant permissions to the `delivery.logs.amazonaws.com` service principal.

```
{
    "Sid": "Allow Logs Delivery to use the key",
    "Effect": "Allow",
    "Principal": {"Service": "delivery.logs.amazonaws.com"},
    "Action": [
        "kms:Encrypt",
```

```
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {"aws:SourceAccount": "111111111111"},
        "ArnLike": {"aws:SourceArn": "arn:aws:logs:us-east-1:111111111111:delivery-
source:*"}
    }
}
```

Configure the source account

After you configure the destination account, follow this procedure to create the delivery source and enable logging for the distribution in the source account.

To configure the source account

1. Create a delivery source for CloudFront standard logging so that you can send log files to CloudWatch Logs.

   You can enter the following AWS CLI command, replacing the name and your distribution ARN.

   ```
   aws logs put-delivery-source --name s3-cf-delivery --resource-arn
    arn:aws:cloudfront::111111111111:distribution/E1TR1RHV123ABC --log-type
    ACCESS_LOGS
   ```

   Output

   ```
   {
       "deliverySource": {
           "name": "s3-cf-delivery",
           "arn": "arn:aws:logs:us-east-1:111111111111:delivery-source:s3-cf-
   delivery",
           "resourceArns":
    ["arn:aws:cloudfront::111111111111:distribution/E1TR1RHV123ABC"],
           "service": "cloudfront",
           "logType": "ACCESS_LOGS"
       }
   ```

```
}
```

2. Create a delivery to map the source account's log delivery source and the destination account's log delivery destination.

   In the following AWS CLI command, specify the delivery destination ARN from the output in Step 1: Configure the destination account.

```
aws logs create-delivery --delivery-source-name s3-cf-delivery --
delivery-destination-arn arn:aws:logs:us-east-1:222222222222:delivery-
destination:cloudfront-delivery-destination
```

Output

```
{
    "delivery": {
        "id": "OPmOpLahVzhx1234",
        "arn": "arn:aws:logs:us-east-1:111111111111:delivery:OPmOpLahVzhx1234",
        "deliverySourceName": "s3-cf-delivery",
        "deliveryDestinationArn": "arn:aws:logs:us-east-1:222222222222:delivery-
destination:cloudfront-delivery-destination",
        "deliveryDestinationType": "S3",
        "recordFields": [
            "date",
            "time",
            "x-edge-location",
            "sc-bytes",
            "c-ip",
            "cs-method",
            "cs(Host)",
            "cs-uri-stem",
            "sc-status",
            "cs(Referer)",
            "cs(User-Agent)",
            "cs-uri-query",
            "cs(Cookie)",
            "x-edge-result-type",
            "x-edge-request-id",
            "x-host-header",
            "cs-protocol",
            "cs-bytes",
            "time-taken",
```

```
            "x-forwarded-for",
            "ssl-protocol",
            "ssl-cipher",
            "x-edge-response-result-type",
            "cs-protocol-version",
            "fle-status",
            "fle-encrypted-fields",
            "c-port",
            "time-to-first-byte",
            "x-edge-detailed-result-type",
            "sc-content-type",
            "sc-content-len",
            "sc-range-start",
            "sc-range-end",
            "c-country",
            "cache-behavior-path-pattern"
        ],
        "fieldDelimiter": "\t"
    }
}
```

3. Verify your cross-account delivery is successful.

    a. From the *source* account, sign in to the CloudFront console and choose your distribution. On the **Logging** tab, under **Type**, you will see an entry created for the S3 cross-account log delivery.

    b. From the *destination* account, sign in to the Amazon S3 console and choose your Amazon S3 bucket. You will see the prefix *MyLogPrefix* in the bucket name and any access logs delivered to that folder.

Output file format

Depending on the delivery destination that you choose, you can specify one of the following formats for log files:

- JSON
- Plain
- w3c
- Raw
- Parquet (Amazon S3 only)

> **Note**
>
> You can only set the output format when you first create the delivery destination. This can't be updated later. To change the output format, delete the delivery and create another one.

For more information, see PutDeliveryDestination in the *Amazon CloudWatch Logs API Reference*.

Edit standard logging settings

You can enable or disable logging and update other log settings by using the CloudFront console or the CloudWatch API. Your changes to logging settings take effect within 12 hours.

For more information, see the following topics:

- To update a distribution by using the CloudFront console, see Update a distribution.
- To update a distribution by using the CloudFront API, see UpdateDistribution in the *Amazon CloudFront API Reference*.
- For more information about CloudWatch Logs API operations, see the Amazon CloudWatch Logs API Reference.

Access log fields

You can select the same log fields that standard logging (legacy) supports. For more information, see log file fields.

In addition, you can select the following real-time access log fields.

1. **`timestamp(ms)`** – Timestamp in milliseconds.
2. **`origin-fbl`** – The number of seconds of first-byte latency between CloudFront and your origin.
3. **`origin-lbl`** – The number of seconds of last-byte latency between CloudFront and your origin.
4. **`asn`** – The autonomous system number (ASN) of the viewer.
5. **`c-country`** – A country code that represents the viewer's geographic location, as determined by the viewer's IP address. For a list of country codes, see ISO 3166-1 alpha-2.
6. **`cache-behavior-path-pattern`** – The path pattern that identifies the cache behavior that matched the viewer request.

7. **`viewer-request-log-data`** – Contains the custom data that your viewer request CloudFront function sends by using the `cf.logCustomData()` helper method. CloudFront URL-encodes the value and truncates it to a maximum of 800 bytes. If no CloudFront function is associated with the viewer request event, or if the function does not call `cf.logCustomData()`, this field is `-`. For more information, see General helper methods.
8. **`viewer-response-log-data`** – Contains the custom data that your viewer response CloudFront function sends by using the `cf.logCustomData()` helper method. CloudFront URL-encodes the value and truncates it to a maximum of 800 bytes. If no CloudFront function is associated with the viewer response event, or if the function does not call `cf.logCustomData()`, this field is `-`. For more information, see General helper methods.
9. **`s-ip`** – The IP address of the CloudFront server that served the request, for example, `192.0.2.183` or `2001:0db8:85a3::8a2e:0370:7334`.
10. **`primary-distribution-id`** – When continuous deployment is enabled, this ID identifies which distribution is the primary in the current distribution.
11. **`primary-distribution-dns-name`** – When continuous deployment is enabled, this value shows the primary domain name that is related to the current CloudFront distribution (for example, d111111abcdef8.cloudfront.net).
12. **`r-host`** – This field is sent for origin requests and it indicates the domain of the origin server used to serve the object. In case of errors, you can use this field to find the last origin attempted, for example: *`cd8jhdejh6a`*`.mediapackagev2.us-east-1.amazonaws.com`.
13. **`sr-reason`** – This field provides a reason why the origin was selected. It's empty when a request to the primary origin succeeds. If origin failover occurs, the field will contain the HTTP error code that led to the failover, such as `Failover:403` or `Failover:502`. In case of origin failover, if the retried request also fails and you have not configured custom error pages, then `r-status` indicates the response of the second origin. However, if you have configured custom error pages along with origin failover, then this will contain the response of the second origin if the request failed and a custom error page was returned instead. If no origin failover occurs but media quality-aware resilience (MQAR) origin selection occurs, then this will be logged as `MediaQuality`. For more information, see Media quality-aware resiliency.
14. **`x-edge-mqcs`** – This field indicates the Media Quality Confidence Score (MQCS) (range: 0 – 100) for media segments that CloudFront retrieved in the CMSD response headers from MediaPackage v2. This field is available for requests matching a cache behavior that has an MQAR-enabled origin group. CloudFront logs this field for media segments that are also served from its cache in addition to origin requests. For more information, see Media quality-aware resiliency.

Common Media Client Data (CMCD) fields

For more information about these fields, see the CTA Specification Web Application Video Ecosystem - Common Media Client Data CTA-5004 document.

- **`cmcd-encoded-bitrate`** – The encoded bitrate of the requested audio or video object.
- **`cmcd-buffer-length`** – The buffer length of the requested media object.
- **`cmcd-buffer-starvation`** – Whether the buffer was starved at some point between the prior request and the object request. This can cause the player to be in a rebuffering state, which can stall the video or audio playback.
- **`cmcd-content-id`** – A unique string that identifies the current content.
- **`cmcd-object-duration`** – The playback duration of the requested object (in milliseconds).
- **`cmcd-deadline`** – The deadline from the request time that the first sample of this object must be available, so that a buffer underrun state or other playback problems are avoided.
- **`cmcd-measured-throughput`** – The throughput between the client and server, as measured by the client.
- **`cmcd-next-object-request`** – The relative path of the next requested object.
- **`cmcd-next-range-request`** – If the next request is a partial object request, this string denotes the byte range to be requested.
- **`cmcd-object-type`** – The media type of the current object being requested.
- **`cmcd-playback-rate`** – 1 if real-time, 2 if double-speed, 0 if not playing.
- **`cmcd-requested-maximum-throughput`** – The requested maximum throughput that the client considers sufficient for asset delivery.
- **`cmcd-streaming-format`** – The streaming format that defines the current request.
- **`cmcd-session-id`** – A GUID identifying the current playback session.
- **`cmcd-stream-type`** – Token identifying segment availability. `v` = all segments are available. `l` = segments become available over time.
- **`cmcd-startup`** – Key is included without a value if the object is needed urgently during startup, seeking, or recovery after a buffer-empty event.
- **`cmcd-top-bitrate`** – The highest bitrate rendition that the client can play.
- **`cmcd-version`** – The version of this specification used for interpreting the defined key names and values. If this key is omitted, the version is 1; according to the specification, both the client and server *must* interpret the version as 1 when this field is omitted.

Send logs to CloudWatch Logs

To send logs to CloudWatch Logs, create or use an existing CloudWatch Logs log group. For more information about configuring a CloudWatch Logs log group, see Working with Log Groups and Log Streams.

After you create your log group, you must have the required permissions to allow standard logging. For more information about the required permissions, see Logs sent to CloudWatch Logs in the *Amazon CloudWatch Logs User Guide*.

Notes

- When you specify the name of the CloudWatch Logs log group, only use the regex pattern `[\w-]`. For more information, see the PutDeliveryDestination API operation in the *Amazon CloudWatch Logs API Reference*.
- Verify that your log group resource policy doesn't exceed the size limit. See the Log group resource policy size limit considerations section in the CloudWatch Logs topic.

Example access log sent to CloudWatch Logs

```
{
"date": "2024-11-14",
"time": "21:34:06",
"x-edge-location": "SOF50-P2",
"asn": "16509",
"timestamp(ms)": "1731620046814",
"origin-fbl": "0.251",
"origin-lbl": "0.251",
"x-host-header": "d111111abcdef8.cloudfront.net",
"cs(Cookie)": "examplecookie=value"
}
```

Send logs to Firehose

To send logs to Firehose, create or use an existing Firehose delivery stream. Then, specify the Firehose delivery stream as the log delivery destination. You must specify a Firehose delivery stream in the US East (N. Virginia) us-east-1 Region.

For information about creating your delivery stream, see Creating an Amazon Data Firehose delivery stream.

After you create your delivery stream, you must have the required permissions to allow standard logging. For more information, see Logs sent to Firehose in the *Amazon CloudWatch Logs User Guide*.

> **Note**
>
> When you specify the name of the Firehose stream, only use the regex pattern `[\w-]`. For more information, see the PutDeliveryDestination API operation in the *Amazon CloudWatch Logs API Reference*.

Example access log sent to Firehose

```
{"date":"2024-11-15","time":"19:45:51","x-edge-location":"SOF50-
P2","asn":"16509","timestamp(ms)":"1731699951183","origin-fbl":"0.254","origin-
lbl":"0.254","x-host-
header":"d111111abcdef8.cloudfront.net","cs(Cookie)":"examplecookie=value"}
{"date":"2024-11-15","time":"19:45:52","x-edge-location":"SOF50-
P2","asn":"16509","timestamp(ms)":"1731699952950","origin-fbl":"0.125","origin-
lbl":"0.125","x-host-
header":"d111111abcdef8.cloudfront.net","cs(Cookie)":"examplecookie=value"}
```

Send logs to Amazon S3

To send your access logs to Amazon S3, create or use an existing S3 bucket. When you enable logging in CloudFront, specify the bucket name. For information about creating a bucket, see Create a bucket in the *Amazon Simple Storage Service User Guide*.

After you create your bucket, you must have the required permissions to allow standard logging. For more information, see Logs sent to Amazon S3 in the *Amazon CloudWatch Logs User Guide*.

- After you enable logging, AWS automatically adds the required bucket policies for you.
- You can also use S3 buckets in the opt-in AWS Regions.

> **Note**
>
> If you already enabled standard logging (legacy) and you want to enable standard logging (v2) to Amazon S3, we recommend that you specify a *different* Amazon S3 bucket or use a *separate path* in the same bucket (for example, use a log prefix or partitioning). This helps you keep track of which log files are associated with which distribution and prevents log files from overwriting each other.

Topics

- Specify an S3 bucket
- Partitioning
- Hive-compatible file name format
- Example paths to access logs
- Example access log sent to Amazon S3

Specify an S3 bucket

When you specify an S3 bucket as the delivery destination, note the following.

The S3 bucket name can only use the regex pattern `[\w-]`. For more information, see the PutDeliveryDestination API operation in the *Amazon CloudWatch Logs API Reference*.

If you specified a prefix for your S3 bucket, your logs appear under that path. If you don't specify a prefix, CloudFront will automatically append the `AWSLogs/{account-id}/CloudFront` prefix for you.

For more information, see Example paths to access logs.

Partitioning

You can use partitioning to organize your access logs when CloudFront sends them to your S3 bucket. This helps you organize and locate your access logs based on the path that you want.

You can use the following variables to create a folder path.

- `{DistributionId}` or `{distributionid}`

- {yyyy}
- {MM}
- {dd}
- {HH}
- {accountid}

You can use any number of variables and specify folder names in your path. CloudFront then uses this path to create a folder structure for you in the S3 bucket.

Examples

- *my_distribution_log_data*/{DistributionId}/*logs*
- /*cloudfront*/{DistributionId}/*my_distribution_log_data*/{yyyy}/{MM}/{dd}/{HH}/*logs*

> **Note**
>
> You can use either variable for distribution ID in the suffix path. However, if you're sending access logs to AWS Glue, you must use the {distributionid} variable because AWS Glue expects partition names to be in lowercase. Update your existing log configuration in CloudFront to replace {DistributionId} with {distributionid}.

Hive-compatible file name format

You can use this option so that S3 objects that contain delivered access logs use a prefix structure that allows for integration with Apache Hive. For more information, see the CreateDelivery API operation.

Example

```
/cloudfront/DistributionId={DistributionId}/my_distribution_log_data/year={yyyy}/
month={MM}/day={dd}/hour={HH}/logs
```

For more information about partitioning and the Hive-compatible options, see the S3DeliveryConfiguration element in the *Amazon CloudWatch Logs API Reference*.

Example paths to access logs

When you specify an S3 bucket as the destination, you can use the following options to create the path to your access logs:

- An Amazon S3 bucket, with or without a prefix
- Partitioning, by using a CloudFront provided variable or entering your own
- Enabling the Hive-compatible option

The following tables show how your access logs appear in your bucket, depending on the options that you choose.

Amazon S3 bucket with a prefix

Amazon S3 bucket name	Partition that you specify in the suffix path	Updated suffix path	Hive-compatible enabled?	Access logs are sent to
`amzn-s3-demo-bucket/MyLogPrefix`	None	None	No	`amzn-s3-demo-bucket/MyLogPrefix/`
`amzn-s3-demo-bucket/MyLogPrefix`	`myFolderA/`	`myFolderA/`	No	`amzn-s3-demo-bucket/MyLogPrefix/myFolderA/`
`amzn-s3-demo-bucket/MyLogPrefix`	`myFolderA/{yyyy}`	`myFolderA/{yyyy}`	Yes	`amzn-s3-demo-bucket/MyLogPrefix/myFolderA/year=2025`

Amazon S3 bucket without a prefix

Amazon S3 bucket name	Partition that you specify in the suffix path	Updated suffix path	Hive-compatible enabled?	Access logs are sent to
amzn-s3-demo-bucket	None	AWSLogs/{account-id}/CloudFront/	No	amzn-s3-demo-bucket/AWSLogs/*<your-account-ID>*/CloudFront/
amzn-s3-demo-bucket	myFolderA/	AWSLogs/{account-id}/CloudFront/myFolderA/	No	amzn-s3-demo-bucket/AWSLogs/*<your-account-ID>*/CloudFront/myFolderA/
amzn-s3-demo-bucket	myFolderA/	AWSLogs/{account-id}/CloudFront/myFolderA/	Yes	amzn-s3-demo-bucket/AWSLogs/aws-account-id=*<your-account-ID>*/CloudFront/myFolderA/
amzn-s3-demo-bucket	myFolderA/{yyyy}	AWSLogs/{account-id}/CloudFront/myFolderA/{yyyy}	Yes	amzn-s3-demo-bucket/AWSLogs/aws-account-id=*<your-acc*

Amazon S3 bucket name	Partition that you specify in the suffix path	Updated suffix path	Hive-comp atible enabled?	Access logs are sent to
				ount-ID> / CloudFront/ myFolderA/ year=2025

AWS account ID as a partition

Amazon S3 bucket name	Partition that you specify in the suffix path	Updated suffix path	Hive-comp atible enabled?	Access logs are sent to
amzn-s3-demo-bucket	None	AWSLogs/{account-id}/CloudFront/	Yes	amzn-s3-demo-bucket/AWSLogs/aws-account-id=*<your-account-ID>* /CloudFront/
amzn-s3-demo-bucket	myFolderA/{accountid}	AWSLogs/{account-id}/CloudFront/myFolderA/{accountid}	Yes	amzn-s3-demo-bucket/AWSLogs/aws-account-id=*<your-account-ID>*/CloudFront/myFolderA/accountid= *<your-account-ID>*

Notes

- The `{account-id}` variable is reserved for CloudFront. CloudFront automatically adds this variable to your suffix path if you specify an Amazon S3 bucket *without* a prefix. If your logs are Hive-compatible, this variable appears as `aws-account-id`.
- You can use the `{accountid}` variable so that CloudFront adds your account ID to the suffix path. If your logs are Hive-compatible, this variable appears as `accountid`.
- For more information about the suffix path, see S3DeliveryConfiguration.

Example access log sent to Amazon S3

```
#Fields: date time x-edge-location asn timestamp(ms) x-host-header cs(Cookie)
2024-11-14    22:30:25    SOF50-P2    16509    1731623425421
d111111abcdef8.cloudfront.net    examplecookie=value2
```

Disable standard logging

You can disable standard logging for your distribution if you no longer need it.

To disable standard logging

1. Sign in to the CloudFront console.
2. Choose **Distribution** and then choose your distribution ID.
3. Choose **Logging** and then under **Access log destinations**, select the destination.
4. Choose **Manage** and then choose **Delete**.
5. Repeat the previous step if you have more than one standard logging.

Note

When you delete standard logging from the CloudFront console, this action only deletes the delivery and the delivery destination. It doesn't delete the delivery source from your AWS account. To delete a delivery source, specify the delivery source name in the `aws logs delete-delivery-source --name DeliverySourceName` command. For more information, see DeleteDeliverySource in the *Amazon CloudWatch Logs API Reference*.

Troubleshoot

Use the following information to fix common issues when you work with CloudFront standard logging (v2).

Delivery source already exists

When you enable standard logging for a distribution, you create a delivery source. You then use that delivery source to create deliveries to the destination type that you want: CloudWatch Logs, Firehose, Amazon S3. Currently, you can only have one delivery source per distribution. If you try to create another delivery source for the same distribution, the following error message appears.

```
This ResourceId has already been used in another Delivery Source in this account
```

To create another delivery source, delete the existing one first. For more information, see DeleteDeliverySource in the *Amazon CloudWatch Logs API Reference*.

I changed the suffix path and the Amazon S3 bucket can't receive my logs

If you enabled standard logging (v2) and specify a bucket ARN without a prefix, CloudFront will append the following default to the suffix path: `AWSLogs/{account-id}/CloudFront`. If you use the CloudFront console or the UpdateDeliveryConfiguration API operation to specify a different suffix path, you must update the Amazon S3 bucket policy to use the same path.

Example: Updating the suffix path

1. Your default suffix path is `AWSLogs/{account-id}/CloudFront` and you replace it with `myFolderA`.
2. Because your new suffix path is different than the path specified in the Amazon S3 bucket policy, your access logs won't be delivered.
3. You can do one of the following steps:
   - Update the Amazon S3 bucket permission from `amzn-s3-demo-bucket/AWSLogs/<your-account-ID>/CloudFront/*` to `amzn-s3-demo-bucket/myFolderA/*`.
   - Update your logging configuration to use the default suffix again: `AWSLogs/{account-id}/CloudFront`

For more information, see Permissions.

Delete log files

CloudFront doesn't automatically delete log files from your destination. For information about deleting log files, see the following topics:

Amazon S3

- Deleting objects in the *Amazon Simple Storage Service Console User Guide*

CloudWatch Logs

- Working with log groups and log streams in the *Amazon CloudWatch Logs User Guide*
- DeleteLogGroup in the *Amazon CloudWatch Logs API Reference*

Firehose

- DeleteDeliveryStream in the *Amazon Data Firehose API Reference*

Pricing

CloudFront doesn't charge for enabling standard logs. However, you can incur charges for the delivery, ingestion, storage or access, depending on the log delivery destination that you select. For more information, see Amazon CloudWatch Logs Pricing. Under **Paid Tier**, choose the **Logs** tab, and then under **Vended Logs**, see the information for each delivery destination.

For more information about pricing for each AWS service, see the following topics:

- Amazon CloudWatch Logs Pricing
- Amazon Data Firehose Pricing
- Amazon S3 Pricing

> **Note**
>
> There are no additional charges for log delivery to Amazon S3, though you incur Amazon S3 charges for storing and accessing the log files. If you enable the **Parquet** option to convert your access logs to Apache Parquet, this option incurs CloudWatch charges. For more information, see the Vended Logs section for CloudWatch pricing.

Configure standard logging (legacy)

Notes

- This topic is for the previous version of standard logging. For the latest version, see Configure standard logging (v2).
- If you already enabled standard logging (legacy) and you want to enable standard logging (v2) to Amazon S3, we recommend that you specify a *different* Amazon S3 bucket or use a *separate path* in the same bucket (for example, use a log prefix or partitioning). This helps you keep track of which log files are associated with which distribution and prevents log files from overwriting each other.

To get started with standard logging (legacy), complete the following steps:

1. Choose an Amazon S3 bucket that will receive your logs and add the required permissions.
2. Configure standard logging (legacy) from the CloudFront console or the CloudFront API. You can only choose an Amazon S3 bucket to receive your logs.
3. View your access logs.

Choose an Amazon S3 bucket for standard logs

When you enable logging for a distribution, you specify the Amazon S3 bucket that you want CloudFront to store log files in. If you're using Amazon S3 as your origin, we recommend that you use a *separate* bucket for your log files.

Specify the Amazon S3 bucket that you want CloudFront to store access logs in, for example, `amzn-s3-demo-bucket.s3.amazonaws.com`.

You can store the log files for multiple distributions in the same bucket. When you enable logging, you can specify an optional prefix for the file names, so you can keep track of which log files are associated with which distributions.

About choosing an S3 bucket

- Your bucket must have access control list (ACL) enabled. If you choose a bucket without ACL enabled from the CloudFront console, an error message will appear. See Permissions.

- Don't choose an Amazon S3 bucket with S3 Object Ownership set to **bucket owner enforced**. That setting disables ACLs for the bucket and the objects in it, which prevents CloudFront from delivering log files to the bucket.
- Legacy logging does not support Amazon S3 buckets in opt-in regions. Please choose a region that is enabled by default or use Standard Logging V2 which does support opt-in regions and additional features. For a list of default and opt-in regions, see AWS Regions.

Permissions

Important

Starting in April 2023, you must enable S3 ACLs for new S3 buckets used for CloudFront standard logs. You can enable ACLs when you create a bucket, or enable ACLs for an existing bucket.
For more information about the changes, see Default settings for new S3 buckets FAQ in the *Amazon Simple Storage Service User Guide* and Heads-Up: Amazon S3 Security Changes Are Coming in April of 2023 in the *AWS News Blog*.

Your AWS account must have the following permissions for the bucket that you specify for log files:

- The ACL for the bucket must grant you `FULL_CONTROL`. If you're the bucket owner, your account has this permission by default. If you're not, the bucket owner must update the ACL for the bucket.
- `s3:GetBucketAcl`
- `s3:PutBucketAcl`

ACL for the bucket

When you create or update a distribution and enable logging, CloudFront uses these permissions to update the ACL for the bucket to give the `awslogsdelivery` account `FULL_CONTROL` permission. The `awslogsdelivery` account writes log files to the bucket. If your account doesn't have the required permissions to update the ACL, creating or updating the distribution will fail.

In some circumstances, if you programmatically submit a request to create a bucket but a bucket with the specified name already exists, S3 resets permissions on the bucket to the default value. If you configured CloudFront to save access logs in an S3 bucket and you stop getting logs in that bucket, check permissions on the bucket to make sure that CloudFront has the necessary permissions.

Restoring the ACL for the bucket

If you remove permissions for the `awslogsdelivery` account, CloudFront won't be able to save logs to the S3 bucket. To enable CloudFront to start saving logs for your distribution again, restore the ACL permission by doing one of the following:

- Disable logging for your distribution in CloudFront, and then enable it again. For more information, see Standard logging.
- Add the ACL permission for `awslogsdelivery` manually by navigating to the S3 bucket in the Amazon S3 console and adding permission. To add the ACL for `awslogsdelivery`, you must provide the canonical ID for the account, which is the following:

  For more information about adding ACLs to S3 buckets, see Configuring ACLs in the *Amazon Simple Storage Service User Guide*.

ACL for each log file

In addition to the ACL on the bucket, there's an ACL on each log file. The bucket owner has `FULL_CONTROL` permission on each log file, the distribution owner (if different from the bucket owner) has no permission, and the `awslogsdelivery` account has read and write permissions.

Disabling logging

If you disable logging, CloudFront doesn't delete the ACLs for either the bucket or the log files. You can delete the ACLs if needed.

Required key policy for SSE-KMS buckets

If the S3 bucket for your standard logs uses server-side encryption with AWS KMS keys (SSE-KMS) by using a customer managed key, you must add the following statement to the key policy for your customer managed key. This allows CloudFront to write log files to the bucket. You can't use SSE-KMS with the AWS managed key because CloudFront won't be able to write log files to the bucket.

```
{
```

```
    "Sid": "Allow CloudFront to use the key to deliver logs",
    "Effect": "Allow",
    "Principal": {
        "Service": "delivery.logs.amazonaws.com"
    },
    "Action": "kms:GenerateDataKey*",
    "Resource": "*"
}
```

If the S3 bucket for your standard logs uses SSE-KMS with an S3 Bucket Key, you also need to add the `kms:Decrypt` permission to the policy statement. In that case, the full policy statement looks like the following.

```
{
    "Sid": "Allow CloudFront to use the key to deliver logs",
    "Effect": "Allow",
    "Principal": {
        "Service": "delivery.logs.amazonaws.com"
    },
    "Action": [
        "kms:GenerateDataKey*",
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

Note

When you enable SSE-KMS for your S3 bucket, specify the complete ARN for the customer managed key. For more information, see Specifying server-side encryption with AWS KMS keys (SSE-KMS) in the *Amazon Simple Storage Service User Guide*.

Enable standard logging (legacy)

To enable standard logs, use the CloudFront console or the CloudFront API.

Contents

- Enable standard logging (legacy) (CloudFront console)
- Enable standard logging (legacy) (CloudFront API)

Enable standard logging (legacy) (CloudFront console)

To enable standard logs for a CloudFront distribution (console)

1. Use the CloudFront console to create a new distribution or update an existing one.
2. For the **Standard logging** section, for **Log delivery**, choose **On**.
3. (Optional) For **Cookie logging**, choose **On** if you want to include cookies in your logs. For more information, see Cookie logging.

   > **Tip**
   >
   > Cookie logging is a global setting that applies to *all* standard logs for your distribution. You can't override this setting for separate delivery destinations.

4. For the **Deliver to** section, specify **Amazon S3 (Legacy)**.
5. Specify your Amazon S3 bucket. If you don't have one already, you can choose **Create** or see the documentation to create a bucket.
6. (Optional) For **Log prefix**, specify the string, if any, that you want CloudFront to prefix to the access log file names for this distribution, for example, `exampleprefix/`. The trailing slash (/) is optional but recommended to simplify browsing your log files. For more information, see Log prefix.
7. Complete the steps to update or create your distribution.
8. From the **Logs** page, verify that the standard logs status is **Enabled** next to the distribution.

   For more information about the standard logging delivery and log fields, see the Standard logging reference.

Enable standard logging (legacy) (CloudFront API)

You can also use the CloudFront API to enable standard logs for your distributions.

To enable standard logs for a distribution (CloudFront API)

- Use the CreateDistribution or UpdateDistribution API operation and configure the LoggingConfig object.

Edit standard logging settings

You can enable or disable logging, change the Amazon S3 bucket where your logs are stored, and change the prefix for log files by using the CloudFront console or the CloudFront API. Your changes to logging settings take effect within 12 hours.

For more information, see the following topics:

- To update a distribution using the CloudFront console, see Update a distribution.
- To update a distribution using the CloudFront API, see UpdateDistribution in the *Amazon CloudFront API Reference*.

Send logs to Amazon S3

When you send your logs to Amazon S3, your logs appear in the following format.

File name format

The name of each log file that CloudFront saves in your Amazon S3 bucket uses the following file name format:

<optional prefix>/*<distribution ID>*.*YYYY*-*MM*-*DD*-*HH*.*unique-ID*.gz

The date and time are in Coordinated Universal Time (UTC).

For example, if you use `example-prefix` as the prefix, and your distribution ID is `EMLARXS9EXAMPLE`, your file names look similar to this:

```
example-prefix/EMLARXS9EXAMPLE.2019-11-14-20.RT4KCN4SGK9.gz
```

When you enable logging for a distribution, you can specify an optional prefix for the file names, so you can keep track of which log files are associated with which distributions. If you include a value for the log file prefix and your prefix doesn't end with a forward slash (/), CloudFront appends one automatically. If your prefix does end with a forward slash, CloudFront doesn't add another one.

The `.gz` at the end of the file name indicates that CloudFront has compressed the log file using gzip.

Standard log file format

Each entry in a log file gives details about a single viewer request. The log files have the following characteristics:

- Use the W3C extended log file format.
- Contain tab-separated values.
- Contain records that are not necessarily in chronological order.
- Contain two header lines: one with the file format version, and another that lists the W3C fields included in each record.
- Contain URL-encoded equivalents for spaces and certain other characters in field values.

  URL-encoded equivalents are used for the following characters:

  - ASCII character codes 0 through 32, inclusive
  - ASCII character codes 127 and higher
  - All characters in the following table

  The URL encoding standard is defined in RFC 1738.

URL-Encoded value	Character
%3C	<
%3E	>
%22	"
%23	#
%25	%
%7B	{
%7D	}
%7C	|
%5C	\
%5E	^
%7E	~

URL-Encoded value	Character
%5B	[
%5D	]
%60	`
%27	'
%20	space

Delete log files

CloudFront doesn't automatically delete log files from your Amazon S3 bucket. For information about deleting log files from an Amazon S3 bucket, see Deleting objects in the *Amazon Simple Storage Service Console User Guide*.

Pricing

Standard logging is an optional feature of CloudFront. CloudFront doesn't charge for enabling standard logs. However, you accrue the usual Amazon S3 charges for storing and accessing the files on Amazon S3. You can delete them at any time.

For more information about Amazon S3 pricing, see Amazon S3 Pricing.

For more information about CloudFront pricing, see CloudFront Pricing.

Standard logging reference

The following sections apply to both standard logging (v2) and standard logging (legacy).

Topics

- Timing of log file delivery
- How requests are logged when the request URL or headers exceed the maximum size
- Log file fields
- Analyze logs

Timing of log file delivery

CloudFront delivers logs for a distribution up to several times an hour. In general, a log file contains information about the requests that CloudFront received during a given time period. CloudFront usually delivers the log file for that time period to your destination within an hour of the events that appear in the log. Note, however, that some or all log file entries for a time period can sometimes be delayed by up to 24 hours. When log entries are delayed, CloudFront saves them in a log file for which the file name includes the date and time of the period in which the requests *occurred*, not the date and time when the file was delivered.

When creating a log file, CloudFront consolidates information for your distribution from all of the edge locations that received requests for your objects during the time period that the log file covers.

CloudFront can save more than one file for a time period depending on how many requests CloudFront receives for the objects associated with a distribution.

CloudFront begins to reliably deliver access logs about four hours after you enable logging. You might get a few access logs before that time.

> **Note**
>
> If no users request your objects during the time period, you don't receive any log files for that period.

How requests are logged when the request URL or headers exceed the maximum size

If the total size of all request headers, including cookies, exceeds 20 KB, or if the URL exceeds 8192 bytes, CloudFront can't parse the request completely and can't log the request. Because the request isn't logged, you won't see in the log files the HTTP error status code returned.

If the request body exceeds the maximum size, the request is logged, including the HTTP error status code.

Log file fields

The log file for a distribution contains 33 fields. The following list contains each field name, in order, along with a description of the information in that field.

1. **`date`**

   The date on which the event occurred in the format `YYYY-MM-DD`. For example, `2019-06-30`. The date and time are in Coordinated Universal Time (UTC). For WebSocket connections, this is the date when the connection closed.

2. **`time`**

   The time when the CloudFront server finished responding to the request (in UTC), for example, `01:42:39`. For WebSocket connections, this is the time when the connection is closed.

3. **`x-edge-location`**

   The edge location that served the request. Each edge location is identified by a three-letter code and an arbitrarily assigned number (for example, DFW3). The three-letter code typically corresponds with the International Air Transport Association (IATA) airport code for an airport near the edge location's geographic location. (These abbreviations might change in the future.)

4. **`sc-bytes`**

   The total number of bytes that the server sent to the viewer in response to the request, including headers. For WebSocket and gRPC connections, this is the total number of bytes sent from the server to the client through the connection.

5. **`c-ip`**

   The IP address of the viewer that made the request, for example, `192.0.2.183` or `2001:0db8:85a3::8a2e:0370:7334`. If the viewer used an HTTP proxy or a load balancer to send the request, the value of this field is the IP address of the proxy or load balancer. See also the `x-forwarded-for` field.

6. **`cs-method`**

   The HTTP request method received from the viewer.

7. **`cs(Host)`**

   The domain name of the CloudFront distribution (for example, d111111abcdef8.cloudfront.net).

8. **`cs-uri-stem`**

   The portion of the request URL that identifies the path and object (for example, `/images/cat.jpg`). Question marks (?) in URLs and query strings are not included in the log.

9. **`sc-status`**

Contains one of the following values:

- The HTTP status code of the server's response (for example, `200`).
- `000`, which indicates that the viewer closed the connection before the server could respond to the request. If the viewer closes the connection after the server starts to send the response, this field contains the HTTP status code of the response that the server started to send.

10 **`cs(Referer)`**

The value of the `Referer` header in the request. This is the name of the domain that originated the request. Common referrers include search engines, other websites that link directly to your objects, and your own website.

11 **`cs(User-Agent)`**

The value of the `User-Agent` header in the request. The `User-Agent` header identifies the source of the request, such as the type of device and browser that submitted the request or, if the request came from a search engine, which search engine.

12 **`cs-uri-query`**

The query string portion of the request URL, if any.

When a URL doesn't contain a query string, this field's value is a hyphen (-). For more information, see Cache content based on query string parameters.

13 **`cs(Cookie)`**

The `Cookie` header in the request, including name—value pairs and the associated attributes.

If you enable cookie logging, CloudFront logs the cookies in all requests regardless of which cookies you choose to forward to the origin. When a request doesn't include a cookie header, this field's value is a hyphen (-). For more information about cookies, see Cache content based on cookies.

14 **`x-edge-result-type`**

How the server classified the response after the last byte left the server. In some cases, the result type can change between the time that the server is ready to send the response and the time that it finishes sending the response. See also the `x-edge-response-result-type` field.

For example, in HTTP streaming, suppose the server finds a segment of the stream in the cache. In that scenario, the value of this field would ordinarily be `Hit`. However, if the viewer closes

the connection before the server has delivered the entire segment, the final result type (and the value of this field) is `Error`.

WebSocket and gRPC connections will have a value of `Miss` for this field because the content is not cacheable and is proxied directly to the origin.

Possible values include:

- `Hit` – The server served the object to the viewer from the cache.
- `RefreshHit` – The server found the object in the cache but the object had expired, so the server contacted the origin to verify that the cache had the latest version of the object.
- `Miss` – The request could not be satisfied by an object in the cache, so the server forwarded the request to the origin and returned the result to the viewer.
- `LimitExceeded` – The request was denied because a CloudFront quota (formerly referred to as a limit) was exceeded.
- `CapacityExceeded` – The server returned an HTTP 503 status code because it didn't have enough capacity at the time of the request to serve the object.
- `Error` – Typically, this means the request resulted in a client error (the value of the `sc-status` field is in the `4xx` range) or a server error (the value of the `sc-status` field is in the `5xx` range). If the value of the `sc-status` field is `200`, or if the value of this field is `Error` and the value of the `x-edge-response-result-type` field is not `Error`, it means the HTTP request was successful but the client disconnected before receiving all of the bytes.
- `Redirect` – The server redirected the viewer from HTTP to HTTPS according to the distribution settings.
- `LambdaExecutionError` – The Lambda@Edge function associated with the distribution didn't complete due to a malformed association, a function timeout, an AWS dependency issue, or another general availability problem.

15**`x-edge-request-id`**

An opaque string that uniquely identifies a request. CloudFront also sends this string in the `x-amz-cf-id` response header.

16**`x-host-header`**

The value that the viewer included in the `Host` header of the request. If you're using the CloudFront domain name in your object URLs (such as d111111abcdef8.cloudfront.net), this field contains that domain name. If you're using alternate domain names (CNAMEs) in your object URLs (such as www.example.com), this field contains the alternate domain name.

If you're using alternate domain names, see `cs(Host)` in field 7 for the domain name that is associated with your distribution.

17 **`cs-protocol`**

The protocol of the viewer request (`http`, `https`, `grpcs`, `ws`, or `wss`).

18 **`cs-bytes`**

The total number of bytes of data that the viewer included in the request, including headers. For WebSocket and gRPC connections, this is the total number of bytes sent from the client to the server on the connection.

19 **`time-taken`**

The number of seconds (to the thousandth of a second, for example, 0.082) from when the server receives the viewer's request to when the server writes the last byte of the response to the output queue, as measured on the server. From the perspective of the viewer, the total time to get the full response will be longer than this value because of network latency and TCP buffering.

20 **`x-forwarded-for`**

If the viewer used an HTTP proxy or a load balancer to send the request, the value of the `c-ip` field is the IP address of the proxy or load balancer. In that case, this field is the IP address of the viewer that originated the request. This field can contain multiple comma-separated IP addresses. Each IP address can be an IPv4 address (for example, `192.0.2.183`) or an IPv6 address (for example, `2001:0db8:85a3::8a2e:0370:7334`).

If the viewer did not use an HTTP proxy or a load balancer, the value of this field is a hyphen (-).

21 **`ssl-protocol`**

When the request used HTTPS, this field contains the SSL/TLS protocol that the viewer and server negotiated for transmitting the request and response. For a list of possible values, see the supported SSL/TLS protocols in Supported protocols and ciphers between viewers and CloudFront.

When `cs-protocol` in field 17 is `http`, the value for this field is a hyphen (-).

22 **`ssl-cipher`**

When the request used HTTPS, this field contains the SSL/TLS cipher that the viewer and server negotiated for encrypting the request and response. For a list of possible values, see the supported SSL/TLS ciphers in Supported protocols and ciphers between viewers and CloudFront.

When `cs-protocol` in field 17 is `http`, the value for this field is a hyphen (-).

23 **`x-edge-response-result-type`**

How the server classified the response just before returning the response to the viewer. See also the `x-edge-result-type` field. Possible values include:

- `Hit` – The server served the object to the viewer from the cache.
- `RefreshHit` – The server found the object in the cache but the object had expired, so the server contacted the origin to verify that the cache had the latest version of the object.
- `Miss` – The request could not be satisfied by an object in the cache, so the server forwarded the request to the origin server and returned the result to the viewer.
- `LimitExceeded` – The request was denied because a CloudFront quota (formerly referred to as a limit) was exceeded.
- `CapacityExceeded` – The server returned a 503 error because it didn't have enough capacity at the time of the request to serve the object.
- `Error` – Typically, this means the request resulted in a client error (the value of the `sc-status` field is in the `4xx` range) or a server error (the value of the `sc-status` field is in the `5xx` range).

  If the value of the `x-edge-result-type` field is `Error` and the value of this field is not `Error`, the client disconnected before finishing the download.
- `Redirect` – The server redirected the viewer from HTTP to HTTPS according to the distribution settings.
- `LambdaExecutionError` – The Lambda@Edge function associated with the distribution didn't complete due to a malformed association, a function timeout, an AWS dependency issue, or another general availability problem.

24 **`cs-protocol-version`**

The HTTP version that the viewer specified in the request. Possible values include `HTTP/0.9`, `HTTP/1.0`, `HTTP/1.1`, `HTTP/2.0`, and `HTTP/3.0`.

25 **`fle-status`**

When field-level encryption is configured for a distribution, this field contains a code that indicates whether the request body was successfully processed. When the server successfully processes the request body, encrypts values in the specified fields, and forwards the request to the origin, the value of this field is `Processed`. The value of `x-edge-result-type` can still indicate a client-side or server-side error in this case.

Possible values for this field include:

- `ForwardedByContentType` – The server forwarded the request to the origin without parsing or encryption because no content type was configured.
- `ForwardedByQueryArgs` – The server forwarded the request to the origin without parsing or encryption because the request contains a query argument that wasn't in the configuration for field-level encryption.
- `ForwardedDueToNoProfile` – The server forwarded the request to the origin without parsing or encryption because no profile was specified in the configuration for field-level encryption.
- `MalformedContentTypeClientError` – The server rejected the request and returned an HTTP 400 status code to the viewer because the value of the `Content-Type` header was in an invalid format.
- `MalformedInputClientError` – The server rejected the request and returned an HTTP 400 status code to the viewer because the request body was in an invalid format.
- `MalformedQueryArgsClientError` – The server rejected the request and returned an HTTP 400 status code to the viewer because a query argument was empty or in an invalid format.
- `RejectedByContentType` – The server rejected the request and returned an HTTP 400 status code to the viewer because no content type was specified in the configuration for field-level encryption.
- `RejectedByQueryArgs` – The server rejected the request and returned an HTTP 400 status code to the viewer because no query argument was specified in the configuration for field-level encryption.
- `ServerError` – The origin server returned an error.

If the request exceeds a field-level encryption quota (formerly referred to as a limit), this field contains one of the following error codes, and the server returns HTTP status code 400 to the viewer. For a list of the current quotas on field-level encryption, see Quotas on field-level encryption.

- `FieldLengthLimitClientError` – A field that is configured to be encrypted exceeded the maximum length allowed.
- `FieldNumberLimitClientError` – A request that the distribution is configured to encrypt contains more than the number of fields allowed.
- `RequestLengthLimitClientError` – The length of the request body exceeded the maximum length allowed when field-level encryption is configured.

If field-level encryption is not configured for the distribution, the value of this field is a hyphen (-).

26 **`fle-encrypted-fields`**

The number of field-level encryption fields that the server encrypted and forwarded to the origin. CloudFront servers stream the processed request to the origin as they encrypt data, so this field can have a value even if the value of `fle-status` is an error.

If field-level encryption is not configured for the distribution, the value of this field is a hyphen (-).

27 **`c-port`**

The port number of the request from the viewer.

28 **`time-to-first-byte`**

The number of seconds between receiving the request and writing the first byte of the response, as measured on the server.

29 **`x-edge-detailed-result-type`**

This field contains the same value as the `x-edge-result-type` field, except in the following cases:

- When the object was served to the viewer from the Origin Shield layer, this field contains `OriginShieldHit`.
- When the object was not in the CloudFront cache and the response was generated by an origin request Lambda@Edge function, this field contains `MissGeneratedResponse`.
- When the value of the `x-edge-result-type` field is `Error`, this field contains one of the following values with more information about the error:
  - `AbortedOrigin` – The server encountered an issue with the origin.

- `ClientCommError` – The response to the viewer was interrupted due to a communication problem between the server and the viewer.
- `ClientGeoBlocked` – The distribution is configured to refuse requests from the viewer's geographic location.
- `ClientHungUpRequest` – The viewer stopped prematurely while sending the request.
- `Error` – An error occurred for which the error type doesn't fit any of the other categories. This error type can occur when the server serves an error response from the cache.
- `InternalError` – An internal CloudFront error occurred. This typically indicates that the error originated within CloudFront infrastructure, not from the origin or the viewer.
- `InvalidRequest` – The server received an invalid request from the viewer.
- `InvalidRequestBlocked` – Access to the requested resource is blocked.
- `InvalidRequestCertificate` – The distribution doesn't match the SSL/TLS certificate for which the HTTPS connection was established.
- `InvalidRequestHeader` – The request contained an invalid header.
- `InvalidRequestMethod` – The distribution is not configured to handle the HTTP request method that was used. This can happen when the distribution supports only cacheable requests.
- `LambdaLimitExceeded` – The request exceeded a Lambda@Edge function limit.
- `OriginCommError` – The request timed out while connecting to the origin, or reading data from the origin.
- `OriginConnectError` – The server couldn't connect to the origin.
- `OriginContentRangeLengthError` – The `Content-Length` header in the origin's response doesn't match the length in the `Content-Range` header.
- `OriginDnsError` – The server couldn't resolve the origin's domain name.
- `OriginError` – The origin returned an incorrect response.
- `OriginHeaderTooBigError` – A header returned by the origin is too big for the edge server to process.
- `OriginInvalidResponseError` – The origin returned an invalid response.
- `OriginReadError` – The server couldn't read from the origin.
- `OriginWriteError` – The server couldn't write to the origin.
- `OriginZeroSizeObjectError` – A zero size object sent from the origin resulted in an error.

- `SlowReaderOriginError` – The viewer was slow to read the message that caused the origin error.

30 **`sc-content-type`**

The value of the HTTP `Content-Type` header of the response.

31 **`sc-content-len`**

The value of the HTTP `Content-Length` header of the response.

32 **`sc-range-start`**

When the response contains the HTTP `Content-Range` header, this field contains the range start value.

33 **`sc-range-end`**

When the response contains the HTTP `Content-Range` header, this field contains the range end value.

34 **`distribution-tenant-id`**

The ID of the distribution tenant.

35 **`connection-id`**

A unique identifier for the TLS connection.

You must enable mTLS for your distributions before you can get information for this field. For more information, see Mutual TLS authentication with CloudFront (Viewer mTLS).

The following is an example log file for a distribution.

```
#Version: 1.0
#Fields: date time x-edge-location sc-bytes c-ip cs-method cs(Host) cs-uri-stem sc-
status cs(Referer) cs(User-Agent) cs-uri-query cs(Cookie) x-edge-result-type x-edge-
request-id x-host-header cs-protocol cs-bytes time-taken x-forwarded-for ssl-protocol
 ssl-cipher x-edge-response-result-type cs-protocol-version fle-status fle-encrypted-
fields c-port time-to-first-byte x-edge-detailed-result-type sc-content-type sc-
content-len sc-range-start sc-range-end
2019-12-04 21:02:31 LAX1 392 192.0.2.100 GET d111111abcdef8.cloudfront.net /
index.html 200 - Mozilla/5.0%20(Windows%20NT%2010.0;%20Win64;
%20x64)%20AppleWebKit/537.36%20(KHTML,%20like
```

```
%20Gecko)%20Chrome/78.0.3904.108%20Safari/537.36 - - Hit
 SOX4xwn4XV6Q4rgb7XiVGOHms_BGlTAC4KyHmureZmBNrjGdRLiNIQ== d111111abcdef8.cloudfront.net
 https 23 0.001 - TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 Hit HTTP/2.0 - - 11040 0.001 Hit
 text/html 78 - -
2019-12-04 21:02:31 LAX1 392 192.0.2.100 GET d111111abcdef8.cloudfront.net /
index.html 200 - Mozilla/5.0%20(Windows%20NT%2010.0;%20Win64;
%20x64)%20AppleWebKit/537.36%20(KHTML,%20like
%20Gecko)%20Chrome/78.0.3904.108%20Safari/537.36 - - Hit
 k6WGMNkEzR5BEM_SaF47gjtX9zBDO2m349OY2an0QPEaUum1ZOLrow== d111111abcdef8.cloudfront.net
 https 23 0.000 - TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 Hit HTTP/2.0 - - 11040 0.000 Hit
 text/html 78 - -
2019-12-04 21:02:31 LAX1 392 192.0.2.100 GET d111111abcdef8.cloudfront.net /
index.html 200 - Mozilla/5.0%20(Windows%20NT%2010.0;%20Win64;
%20x64)%20AppleWebKit/537.36%20(KHTML,%20like
%20Gecko)%20Chrome/78.0.3904.108%20Safari/537.36 - - Hit
 f37nTMVvnKvV2ZSvEsivup_c2kZ7VXzYdjC-GUQZ5qNs-89BlWazbw== d111111abcdef8.cloudfront.net
 https 23 0.001 - TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 Hit HTTP/2.0 - - 11040 0.001 Hit
 text/html 78 - -
2019-12-13 22:36:27 SEA19-C1 900 192.0.2.200 GET d111111abcdef8.cloudfront.net /
favicon.ico 502 http://www.example.com/ Mozilla/5.0%20(Windows
%20NT%2010.0;%20Win64;%20x64)%20AppleWebKit/537.36%20(KHTML,
%20like%20Gecko)%20Chrome/78.0.3904.108%20Safari/537.36 - - Error
 1pkpNfBQ39sYMnjjUQjmH2w1wdJnbHYTbag21o_3OfcQgPzdL2RSSQ== www.example.com http 675
 0.102 - - - Error HTTP/1.1 - - 25260 0.102 OriginDnsError text/html 507 - -
2019-12-13 22:36:26 SEA19-C1 900 192.0.2.200 GET d111111abcdef8.cloudfront.net / 502
 - Mozilla/5.0%20(Windows%20NT%2010.0;%20Win64;%20x64)%20AppleWebKit/537.36%20(KHTML,
%20like%20Gecko)%20Chrome/78.0.3904.108%20Safari/537.36 - - Error
 3AqrZGCnF_g0-5KOvfA7c9XLcf4YGvMFSeFdIetR1N_2y8jSis8Zxg== www.example.com http 735
 0.107 - - - Error HTTP/1.1 - - 3802 0.107 OriginDnsError text/html 507 - -
2019-12-13 22:37:02 SEA19-C2 900 192.0.2.200 GET d111111abcdef8.cloudfront.net / 502
 - curl/7.55.1 - - Error kBkDzGnceVtWHqSCqBUqtA_cEs2T3tFUBbnBNkB9El_uVRhHgcZfcw==
 www.example.com http 387 0.103 - - - Error HTTP/1.1 - - 12644 0.103 OriginDnsError
 text/html 507 - -
```

Analyze logs

Because you can receive multiple access logs per hour, we recommend that you combine all the log files you receive for a given time period into one file. You can then analyze the data for that period more accurately and completely.

One way to analyze your access logs is to use Amazon Athena. Athena is an interactive query service that can help you analyze data for AWS services, including CloudFront. To learn more, see Querying Amazon CloudFront Logs in the *Amazon Athena User Guide*.

In addition, the following AWS blog posts discuss some ways to analyze access logs.

- Amazon CloudFront Request Logging (for content delivered by using HTTP)
- Enhanced CloudFront Logs, Now With Query Strings

Use real-time access logs

With CloudFront real-time access logs, you can get information about requests made to a distribution in real time (logs are delivered within seconds of receiving the requests). You can use real-time access logs to monitor, analyze, and take action based on content delivery performance.

CloudFront real-time access logs are configurable. You can choose:

- The *sampling rate* for your real-time logs—that is, the percentage of requests for which you want to receive real-time access log records.
- The specific fields that you want to receive in the log records.
- The specific cache behaviors (path patterns) that you want to receive real-time logs for.

CloudFront real-time access logs are delivered to the data stream of your choice in Amazon Kinesis Data Streams. You can build your own Kinesis data stream consumer, or use Amazon Data Firehose to send the log data to Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service (OpenSearch Service), or a third-party log processing service.

CloudFront charges for real-time access logs, in addition to the charges you incur for using Kinesis Data Streams. For more information about pricing, see Amazon CloudFront Pricing and Amazon Kinesis Data Streams pricing.

> **Important**
>
> We recommend that you use the logs to understand the nature of the requests for your content, not as a complete accounting of all requests. CloudFront delivers real-time access logs on a best-effort basis. The log entry for a particular request might be delivered long after the request was actually processed and, in rare cases, a log entry might not be delivered at all. When a log entry is omitted from real-time access logs, the number of entries in the real-time access logs won't match the usage that appears in the AWS billing and usage reports.

Topics

- Create and use real-time access log configurations
- Understand real-time access log configurations
- Create a Kinesis Data Streams consumer
- Troubleshoot real-time access logs

Create and use real-time access log configurations

To get information about requests made to a distribution in real time, you can use a real-time access log configurations. Logs are delivered within seconds of receiving the requests. You can create a real-time access log configuration in the CloudFront console, with the AWS Command Line Interface (AWS CLI), or with the CloudFront API.

To use a real-time access log configuration, you attach it to one or more cache behaviors in a CloudFront distribution.

Console

To create a real-time access log configuration

1. Sign in to the AWS Management Console and open the **Logs** page in the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home?#/logs.
2. Choose the **Real-time configurations** tab.
3. Choose **Create configuration**.
4. For **Name**, enter a name for the configuration.
5. For **Sampling rate**, enter the percentage of requests for which you want to receive log records.
6. For **Fields**, choose the fields to receive in the real-time access logs.
   - To include all CMCD fields for your logs, choose **CMCD all keys**.
7. For **Endpoint**, choose one or more Kinesis data streams to receive real-time access logs.

> **Note**
>
> CloudFront real-time access logs are delivered to the data stream that you specify in Kinesis Data Streams. To read and analyze your real-time access logs, you can

build your own Kinesis data stream consumer. You can also use Firehose to send the log data to Amazon S3, Amazon Redshift, Amazon OpenSearch Service, or a third-party log processing service.

8. For **IAM role**, choose **Create new service role** or choose an existing role. You must have permission to create IAM roles.
9. (Optional) For **Distribution**, choose a CloudFront distribution and cache behavior to attach to the real-time access log configuration.
10. Choose **Create configuration**.

If successful, the console shows the details of the real-time access log configuration that you just created.

For more information, see Understand real-time access log configurations.

AWS CLI

To create a real-time access log configuration with the AWS CLI, use the **aws cloudfront create-realtime-log-config** command. You can use an input file to provide the command's input parameters, rather than specifying each individual parameter as command line input.

To create a real-time access log configuration (CLI with input file)

1. Use the following command to create a file named `rtl-config.yaml` that contains all of the input parameters for the **create-realtime-log-config** command.

```
aws cloudfront create-realtime-log-config --generate-cli-skeleton yaml-input >
 rtl-config.yaml
```

2. Open the file named `rtl-config.yaml` that you just created. Edit the file to specify the real-time access log configuration settings that you want, then save the file. Note the following:

   - For `StreamType`, the only valid value is `Kinesis`.

   For more information about the real-time log configuration settings, see Understand real-time access log configurations.

3. Use the following command to create the real-time access log configuration using input parameters from the `rtl-config.yaml` file.

```
aws cloudfront create-realtime-log-config --cli-input-yaml file://rtl-
config.yaml
```

If successful, the command's output shows the details of the real-time access log configuration that you just created.

To attach a real-time access log configuration to an existing distribution (CLI with input file)

1. Use the following command to save the distribution configuration for the CloudFront distribution that you want to update. Replace *distribution_ID* with the distribution's ID.

```
aws cloudfront get-distribution-config --id distribution_ID --output yaml >
 dist-config.yaml
```

2. Open the file named `dist-config.yaml` that you just created. Edit the file, making the following changes to each cache behavior that you are updating to use a real-time access log configuration.

   - In the cache behavior, add a field named `RealtimeLogConfigArn`. For the field's value, use the ARN of the real-time access log configuration that you want to attach to this cache behavior.
   - Rename the `ETag` field to `IfMatch`, but don't change the field's value.

   Save the file when finished.

3. Use the following command to update the distribution to use the real-time access log configuration. Replace *distribution_ID* with the distribution's ID.

```
aws cloudfront update-distribution --id distribution_ID --cli-input-yaml file://
dist-config.yaml
```

If successful, the command's output shows the details of the distribution that you just updated.

API

To create a real-time access log configuration with the CloudFront API, use the CreateRealtimeLogConfig API operation. For more information about the parameters that you specify in this API call, see Understand real-time access log configurations and the API reference documentation for your AWS SDK or other API client.

After you create a real-time access log configuration, you can attach it to a cache behavior, by using one of the following API operations:

- To attach it to a cache behavior in an existing distribution, use UpdateDistribution.
- To attach it to a cache behavior in a new distribution, use CreateDistribution.

For both of these API operations, provide the ARN of the real-time access log configuration in the `RealtimeLogConfigArn` field, inside a cache behavior. For more information about the other fields that you specify in these API calls, see All distribution settings reference and the API reference documentation for your AWS SDK or other API client.

Understand real-time access log configurations

To use CloudFront real-time access logs, you start by creating a real-time access log configuration. The real-time access log configuration contains information about which log fields you want to receive, the *sampling rate* for log records, and the Kinesis data stream where you want to deliver the logs.

Specifically, a real-time access log configuration contains the following settings:

Contents

- Name
- Sampling rate
- Fields
- Endpoint (Kinesis Data Streams)
- IAM role

Name

A name to identify the real-time access log configuration.

Sampling rate

The sampling rate is a whole number between 1 and 100 (inclusive) that determines the percentage of viewer requests that are sent to Kinesis Data Streams as real-time access log records. To include every viewer request in your real-time access logs, specify 100 for the sampling rate. You might choose a lower sampling rate to reduce costs while still receiving a representative sample of request data in your real-time access logs.

Fields

A list of the fields that are included in each real-time access log record. Each log record can contain up to 40 fields, and you can choose to receive all of the available fields, or only the fields that you need for monitoring and analyzing performance.

The following list contains each field name and a description of the information in that field. The fields are listed in the order in which they appear in the log records that are delivered to Kinesis Data Streams.

Fields 46-63 are common media client data (CMCD) that media player clients can send to CDNs with each request. You can use this data to understand each request, such as the media type (audio, video), playback rate, and streaming length. These fields will only appear in your real-time access logs if they're sent to CloudFront.

1. **`timestamp`**

   The date and time at which the edge server finished responding to the request.

2. **`c-ip`**

   The IP address of the viewer that made the request, for example, `192.0.2.183` or `2001:0db8:85a3::8a2e:0370:7334`. If the viewer used an HTTP proxy or a load balancer to send the request, the value of this field is the IP address of the proxy or load balancer. See also the `x-forwarded-for` field.

3. **`s-ip`**

   The IP address of the CloudFront server that served the request, for example, `192.0.2.183` or `2001:0db8:85a3::8a2e:0370:7334`.

4. **`time-to-first-byte`**

   The number of seconds between receiving the request and writing the first byte of the response, as measured on the server.

5. **`sc-status`**

   The HTTP status code of the server's response (for example, `200`).

6. **`sc-bytes`**

   The total number of bytes that the server sent to the viewer in response to the request, including headers. For WebSocket and gRPC connections, this is the total number of bytes sent from the server to the client through the connection.

7. **`cs-method`**

   The HTTP request method received from the viewer.

8. **`cs-protocol`**

   The protocol of the viewer request (`http`, `https`, `grpcs`, `ws`, or `wss`).

9. **`cs-host`**

   The value that the viewer included in the `Host` header of the request. If you're using the CloudFront domain name in your object URLs (such as d111111abcdef8.cloudfront.net), this field contains that domain name. If you're using alternate domain names (CNAMEs) in your object URLs (such as www.example.com), this field contains the alternate domain name.

10. **`cs-uri-stem`**

    The entire request URL, including the query string (if one exists), but without the domain name. For example, `/images/cat.jpg?mobile=true`.

    > **Note**
    >
    > In standard logs, the `cs-uri-stem` value doesn't include the query string.

11. **`cs-bytes`**

    The total number of bytes of data that the viewer included in the request, including headers. For WebSocket and gRPC connections, this is the total number of bytes sent from the client to the server on the connection.

12. **`x-edge-location`**

    The edge location that served the request. Each edge location is identified by a three-letter code and an arbitrarily assigned number (for example, DFW3). The three-letter code typically

corresponds with the International Air Transport Association (IATA) airport code for an airport near the edge location's geographic location. (These abbreviations might change in the future.)

13 **x-edge-request-id**

An opaque string that uniquely identifies a request. CloudFront also sends this string in the `x-amz-cf-id` response header.

14 **x-host-header**

The domain name of the CloudFront distribution (for example, d111111abcdef8.cloudfront.net).

15 **time-taken**

The number of seconds (to the thousandth of a second, for example, 0.082) from when the server receives the viewer's request to when the server writes the last byte of the response to the output queue, as measured on the server. From the perspective of the viewer, the total time to get the full response will be longer than this value because of network latency and TCP buffering.

16 **cs-protocol-version**

The HTTP version that the viewer specified in the request. Possible values include `HTTP/0.9`, `HTTP/1.0`, `HTTP/1.1`, `HTTP/2.0`, and `HTTP/3.0`.

17 **c-ip-version**

The IP version of the request (IPv4 or IPv6).

18 **cs-user-agent**

The value of the `User-Agent` header in the request. The `User-Agent` header identifies the source of the request, such as the type of device and browser that submitted the request or, if the request came from a search engine, which search engine.

19 **cs-referer**

The value of the `Referer` header in the request. This is the name of the domain that originated the request. Common referrers include search engines, other websites that link directly to your objects, and your own website.

20 **cs-cookie**

The `Cookie` header in the request, including name—value pairs and the associated attributes.

> **Note**
>
> This field is truncated to 800 bytes.

21 **`cs-uri-query`**

The query string portion of the request URL, if any.

22 **`x-edge-response-result-type`**

How the server classified the response just before returning the response to the viewer. See also the `x-edge-result-type` field. Possible values include:

- `Hit` – The server served the object to the viewer from the cache.
- `RefreshHit` – The server found the object in the cache but the object had expired, so the server contacted the origin to verify that the cache had the latest version of the object.
- `Miss` – The request could not be satisfied by an object in the cache, so the server forwarded the request to the origin server and returned the result to the viewer.
- `LimitExceeded` – The request was denied because a CloudFront quota (formerly referred to as a limit) was exceeded.
- `CapacityExceeded` – The server returned a 503 error because it didn't have enough capacity at the time of the request to serve the object.
- `Error` – Typically, this means the request resulted in a client error (the value of the `sc-status` field is in the `4xx` range) or a server error (the value of the `sc-status` field is in the `5xx` range).

  If the value of the `x-edge-result-type` field is `Error` and the value of this field is not `Error`, the client disconnected before finishing the download.
- `Redirect` – The server redirected the viewer from HTTP to HTTPS according to the distribution settings.
- `LambdaExecutionError` – The Lambda@Edge function associated with the distribution didn't complete due to a malformed association, a function timeout, an AWS dependency issue, or another general availability problem.

23 **`x-forwarded-for`**

If the viewer used an HTTP proxy or a load balancer to send the request, the value of the `c-ip` field is the IP address of the proxy or load balancer. In that case, this field is the IP address

of the viewer that originated the request. This field can contain multiple comma-separated IP addresses. Each IP address can be an IPv4 address (for example, `192.0.2.183`) or an IPv6 address (for example, `2001:0db8:85a3::8a2e:0370:7334`).

24 **`ssl-protocol`**

When the request used HTTPS, this field contains the SSL/TLS protocol that the viewer and server negotiated for transmitting the request and response. For a list of possible values, see the supported SSL/TLS protocols in Supported protocols and ciphers between viewers and CloudFront.

25 **`ssl-cipher`**

When the request used HTTPS, this field contains the SSL/TLS cipher that the viewer and server negotiated for encrypting the request and response. For a list of possible values, see the supported SSL/TLS ciphers in Supported protocols and ciphers between viewers and CloudFront.

26 **`x-edge-result-type`**

How the server classified the response after the last byte left the server. In some cases, the result type can change between the time that the server is ready to send the response and the time that it finishes sending the response. See also the `x-edge-response-result-type` field.

For example, in HTTP streaming, suppose the server finds a segment of the stream in the cache. In that scenario, the value of this field would ordinarily be `Hit`. However, if the viewer closes the connection before the server has delivered the entire segment, the final result type (and the value of this field) is `Error`.

WebSocket and gRPC connections will have a value of `Miss` for this field because the content is not cacheable and is proxied directly to the origin.

Possible values include:

- `Hit` – The server served the object to the viewer from the cache.
- `RefreshHit` – The server found the object in the cache but the object had expired, so the server contacted the origin to verify that the cache had the latest version of the object.
- `Miss` – The request could not be satisfied by an object in the cache, so the server forwarded the request to the origin and returned the result to the viewer.
- `LimitExceeded` – The request was denied because a CloudFront quota (formerly referred to as a limit) was exceeded.

- `CapacityExceeded` – The server returned an HTTP 503 status code because it didn't have enough capacity at the time of the request to serve the object.
- `Error` – Typically, this means the request resulted in a client error (the value of the `sc-status` field is in the `4xx` range) or a server error (the value of the `sc-status` field is in the `5xx` range). If the value of the `sc-status` field is `200`, or if the value of this field is `Error` and the value of the `x-edge-response-result-type` field is not `Error`, it means the HTTP request was successful but the client disconnected before receiving all of the bytes.
- `Redirect` – The server redirected the viewer from HTTP to HTTPS according to the distribution settings.
- `LambdaExecutionError` – The Lambda@Edge function associated with the distribution didn't complete due to a malformed association, a function timeout, an AWS dependency issue, or another general availability problem.

27. **`fle-encrypted-fields`**

    The number of field-level encryption fields that the server encrypted and forwarded to the origin. CloudFront servers stream the processed request to the origin as they encrypt data, so this field can have a value even if the value of `fle-status` is an error.

28. **`fle-status`**

    When field-level encryption is configured for a distribution, this field contains a code that indicates whether the request body was successfully processed. When the server successfully processes the request body, encrypts values in the specified fields, and forwards the request to the origin, the value of this field is `Processed`. The value of `x-edge-result-type` can still indicate a client-side or server-side error in this case.

    Possible values for this field include:

    - `ForwardedByContentType` – The server forwarded the request to the origin without parsing or encryption because no content type was configured.
    - `ForwardedByQueryArgs` – The server forwarded the request to the origin without parsing or encryption because the request contains a query argument that wasn't in the configuration for field-level encryption.
    - `ForwardedDueToNoProfile` – The server forwarded the request to the origin without parsing or encryption because no profile was specified in the configuration for field-level encryption.

- `MalformedContentTypeClientError` – The server rejected the request and returned an HTTP 400 status code to the viewer because the value of the `Content-Type` header was in an invalid format.
- `MalformedInputClientError` – The server rejected the request and returned an HTTP 400 status code to the viewer because the request body was in an invalid format.
- `MalformedQueryArgsClientError` – The server rejected the request and returned an HTTP 400 status code to the viewer because a query argument was empty or in an invalid format.
- `RejectedByContentType` – The server rejected the request and returned an HTTP 400 status code to the viewer because no content type was specified in the configuration for field-level encryption.
- `RejectedByQueryArgs` – The server rejected the request and returned an HTTP 400 status code to the viewer because no query argument was specified in the configuration for field-level encryption.
- `ServerError` – The origin server returned an error.

If the request exceeds a field-level encryption quota (formerly referred to as a limit), this field contains one of the following error codes, and the server returns HTTP status code 400 to the viewer. For a list of the current quotas on field-level encryption, see Quotas on field-level encryption.

- `FieldLengthLimitClientError` – A field that is configured to be encrypted exceeded the maximum length allowed.
- `FieldNumberLimitClientError` – A request that the distribution is configured to encrypt contains more than the number of fields allowed.
- `RequestLengthLimitClientError` – The length of the request body exceeded the maximum length allowed when field-level encryption is configured.

29 **`sc-content-type`**

The value of the HTTP `Content-Type` header of the response.

30 **`sc-content-len`**

The value of the HTTP `Content-Length` header of the response.

31 **`sc-range-start`**

When the response contains the HTTP `Content-Range` header, this field contains the range start value.

32**sc-range-end**

When the response contains the HTTP `Content-Range` header, this field contains the range end value.

33**c-port**

The port number of the request from the viewer.

34**x-edge-detailed-result-type**

This field contains the same value as the `x-edge-result-type` field, except in the following cases:

- When the object was served to the viewer from the Origin Shield layer, this field contains `OriginShieldHit`.
- When the object was not in the CloudFront cache and the response was generated by an origin request Lambda@Edge function, this field contains `MissGeneratedResponse`.
- When the value of the `x-edge-result-type` field is `Error`, this field contains one of the following values with more information about the error:
  - `AbortedOrigin` – The server encountered an issue with the origin.
  - `ClientCommError` – The response to the viewer was interrupted due to a communication problem between the server and the viewer.
  - `ClientGeoBlocked` – The distribution is configured to refuse requests from the viewer's geographic location.
  - `ClientHungUpRequest` – The viewer stopped prematurely while sending the request.
  - `Error` – An error occurred for which the error type doesn't fit any of the other categories. This error type can occur when the server serves an error response from the cache.
  - `InternalError` – An internal CloudFront error occurred. This typically indicates that the error originated within CloudFront infrastructure, not from the origin or the viewer.
  - `InvalidRequest` – The server received an invalid request from the viewer.
  - `InvalidRequestBlocked` – Access to the requested resource is blocked.
  - `InvalidRequestCertificate` – The distribution doesn't match the SSL/TLS certificate for which the HTTPS connection was established.
  - `InvalidRequestHeader` – The request contained an invalid header.

- `InvalidRequestMethod` – The distribution is not configured to handle the HTTP request method that was used. This can happen when the distribution supports only cacheable requests.
- `LambdaLimitExceeded` – The request exceeded a Lambda@Edge function limit.
- `OriginCommError` – The request timed out while connecting to the origin, or reading data from the origin.
- `OriginConnectError` – The server couldn't connect to the origin.
- `OriginContentRangeLengthError` – The `Content-Length` header in the origin's response doesn't match the length in the `Content-Range` header.
- `OriginDnsError` – The server couldn't resolve the origin's domain name.
- `OriginError` – The origin returned an incorrect response.
- `OriginHeaderTooBigError` – A header returned by the origin is too big for the edge server to process.
- `OriginInvalidResponseError` – The origin returned an invalid response.
- `OriginReadError` – The server couldn't read from the origin.
- `OriginWriteError` – The server couldn't write to the origin.
- `OriginZeroSizeObjectError` – A zero size object sent from the origin resulted in an error.
- `SlowReaderOriginError` – The viewer was slow to read the message that caused the origin error.

35**`c-country`**

A country code that represents the viewer's geographic location, as determined by the viewer's IP address. For a list of country codes, see ISO 3166-1 alpha-2.

36**`cs-accept-encoding`**

The value of the `Accept-Encoding` header in the viewer request.

37**`cs-accept`**

The value of the `Accept` header in the viewer request.

38**`cache-behavior-path-pattern`**

The path pattern that identifies the cache behavior that matched the viewer request.

39**`cs-headers`**

The HTTP headers (names and values) in the viewer request.

> **Note**
>
> This field is truncated to 800 bytes.

40 **`cs-header-names`**

The names of the HTTP headers (not values) in the viewer request.

> **Note**
>
> This field is truncated to 800 bytes.

41 **`cs-headers-count`**

The number of HTTP headers in the viewer request.

42 **`primary-distribution-id`**

When continuous deployment is enabled, this ID identifies which distribution is the primary in the current distribution.

43 **`primary-distribution-dns-name`**

When continuous deployment is enabled, this value shows the primary domain name that is related to the current CloudFront distribution (for example, d111111abcdef8.cloudfront.net).

44 **`origin-fbl`**

The number of seconds of first-byte latency between CloudFront and your origin.

45 **`origin-lbl`**

The number of seconds of last-byte latency between CloudFront and your origin.

46 **`asn`**

The autonomous system number (ASN) of the viewer.

47.

CMCD fields in real-time access logs

For more information about these fields, see the CTA Specification Web Application Video Ecosystem - Common Media Client Data CTA-5004 document.

48 **`cmcd-encoded-bitrate`**

The encoded bitrate of the requested audio or video object.

49 **`cmcd-buffer-length`**

The buffer length of the requested media object.

50 **`cmcd-buffer-starvation`**

Whether the buffer was starved at some point between the prior request and the object request. This can cause the player to be in a rebuffering state, which can stall the video or audio playback.

51 **`cmcd-content-id`**

A unique string that identifies the current content.

52 **`cmcd-object-duration`**

The playback duration of the requested object (in milliseconds).

53 **`cmcd-deadline`**

The deadline from the request time that the first sample of this object must be available, so that a buffer underrun state or other playback problems are avoided.

54 **`cmcd-measured-throughput`**

The throughput between the client and server, as measured by the client.

55 **`cmcd-next-object-request`**

The relative path of the next requested object.

56 **`cmcd-next-range-request`**

If the next request is a partial object request, this string denotes the byte range to be requested.

57 **`cmcd-object-type`**

The media type of the current object being requested.

58**cmcd-playback-rate**

1 if real-time, 2 if double-speed, 0 if not playing.

59**cmcd-requested-maximum-throughput**

The requested maximum throughput that the client considers sufficient for asset delivery.

60**cmcd-streaming-format**

The streaming format that defines the current request.

61**cmcd-session-id**

A GUID identifying the current playback session.

62**cmcd-stream-type**

Token identifying segment availability. `v` = all segments are available. `l` = segments become available over time.

63**cmcd-startup**

Key is included without a value if the object is needed urgently during startup, seeking, or recovery after a buffer-empty event.

64**cmcd-top-bitrate**

The highest bitrate rendition that the client can play.

65**cmcd-version**

The version of this specification used for interpreting the defined key names and values. If this key is omitted, the version is 1; according to the specification, both the client and server *must* interpret the version as 1 when this field is omitted.

66**r-host**

This field is sent for origin requests and it indicates the domain of the origin server used to serve the object. In case of errors, you can use this field to find the last origin attempted, for example: *cd8jhdejh6a*`.mediapackagev2.us-east-1.amazonaws.com`.

67**sr-reason**

This field provides a reason why the origin was selected. It's empty when a request to the primary origin succeeds.

If origin failover occurs, the field will contain the HTTP error code that led to the failover, such as `Failover:403` or `Failover:502`. In case of origin failover, if the retried request also fails and you have not configured custom error pages, then `r-status` indicates the response of the second origin. However, if you have configured custom error pages along with origin failover, then this will contain the response of the second origin if the request failed and a custom error page was returned instead.

If no origin failover occurs but media quality-aware resilience (MQAR) origin selection occurs, then this will be logged as `MediaQuality`. For more information, see Media quality-aware resiliency.

68 **`x-edge-mqcs`**

This field indicates the Media Quality Confidence Score (MQCS) (range: 0 – 100) for media segments that CloudFront retrieved in the CMSD response headers from MediaPackage v2. This field is available for requests matching a cache behavior that has an MQAR-enabled origin group. CloudFront logs this field for media segments that are also served from its cache in addition to origin requests. For more information, see Media quality-aware resiliency.

69 **`distribution-tenant-id`**

The ID of the distribution tenant.

70 **`connection-id`**

A unique identifier for the TLS connection.

You must enable mTLS for your distributions before you can get information for this field. For more information, see Mutual TLS authentication with CloudFront (Viewer mTLS).

71 **`viewer-request-log-data`**

Custom data that your viewer request CloudFront function sends by using the `cf.logCustomData()` helper method. This field contains the string value that you passed to the method during the viewer request event. CloudFront URL-encodes the value and truncates it to a maximum of 800 bytes. If no CloudFront function is associated with the viewer request event, or if the function does not call `cf.logCustomData()`, this field is `-`. For more information, see General helper methods.

72**viewer-response-log-data**

Custom data that your viewer response CloudFront function sends by using the `cf.logCustomData()` helper method. This field contains the string value that you passed to the method during the viewer response event. CloudFront URL-encodes the value and truncates it to a maximum of 800 bytes. If no CloudFront function is associated with the viewer response event, or if the function does not call `cf.logCustomData()`, this field is `-`. For more information, see General helper methods.

Endpoint (Kinesis Data Streams)

The endpoint contains information about the Kinesis Data Streams where you want to send real-time logs. You provide the Amazon Resource Name (ARN) of the data stream.

For more information about creating a Kinesis Data Streams, see the following topics in the *Amazon Kinesis Data Streams Developer Guide*.

- Creating and managing streams
- Perform basic Kinesis Data Streams operations using the AWS CLI
- Creating a stream (uses the AWS SDK for Java)

When you create a data stream, you need to specify the number of shards. Use the following information to help you estimate the number of shards you need.

To estimate the number of shards for your Kinesis data stream

1. Calculate (or estimate) the number of requests per second that your CloudFront distribution receives.

   You can use the CloudFront usage reports (in the CloudFront console) and the CloudFront metrics (in the CloudFront and Amazon CloudWatch consoles) to help you calculate your requests per second.

2. Determine the typical size of a single real-time access log record.

   In general, a single log record is around 500 bytes. A large record that includes all available fields is generally around 1 KB.

If you're not sure what your log record size is, you can enable real-time logs with a low sampling rate (for example, 1%), and then calculate the average record size using monitoring data in Kinesis Data Streams (total incoming bytes divided by total number of records).

3. On the Amazon Kinesis Data Streams pricing page, under AWS Pricing Calculator, choose **Create your custom estimate now**.

   - In the calculator, enter the number of requests (records) per second.
   - Enter the average record size of a single log record.
   - Choose **Show calculations**.

   The pricing calculator shows you the number of shards you need and the estimated cost.

IAM role

The AWS Identity and Access Management (IAM) role that gives CloudFront permission to deliver real-time access logs to your Kinesis data stream.

When you create a real-time access log configuration with the CloudFront console, you can choose **Create new service role** to let the console create the IAM role for you.

When you create a real-time access log configuration with AWS CloudFormation or the CloudFront API (AWS CLI or SDK), you must create the IAM role yourself and provide the role ARN. To create the IAM role yourself, use the following policies.

IAM role trust policy

To use the following IAM role trust policy, replace *111122223333* with your AWS account number. The `Condition` element in this policy helps to prevent the confused deputy problem because CloudFront can only assume this role on behalf of a distribution in your AWS account.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

IAM role permissions policy for an unencrypted data stream

To use the following policy, replace *arn:aws:kinesis:eusc-de-east-1:123456789012:stream/StreamName* with the ARN of your Kinesis data stream.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws-eusc:kinesis:eusc-de-east-1:123456789012:stream/
StreamName"
            ]
        }
    ]
}
```

IAM role permissions policy for an encrypted data stream

To use the following policy, replace *arn:aws:kinesis:eusc-de-east-1:123456789012:stream/StreamName* with the ARN of your Kinesis data stream and *arn:aws:kms:eusc-de-east-1:123456789012:key/e58a3d0b-fe4f-4047-a495-ae03cc73d486* with the ARN of your AWS KMS key.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws-eusc:kinesis:eusc-de-east-1:123456789012:stream/
StreamName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": [
                "arn:aws-eusc:kms:eusc-de-east-1:123456789012:key/e58a3d0b-
fe4f-4047-a495-ae03cc73d486"
            ]
        }
    ]
}
```

Create a Kinesis Data Streams consumer

To read and analyze your real-time access logs, you build or use a Kinesis Data Streams *consumer*. When you build a consumer for CloudFront real-time logs, it's important to know that the fields in every real-time access log record are always delivered in the same order, as listed in the Fields section. Make sure that you build your consumer to accommodate this fixed order.

For example, consider a real-time access log configuration that includes only these three fields: `time-to-first-byte`, `sc-status`, and `c-country`. In this scenario, the last field, `c-country`, is always field number 3 in every log record. However, if you later add fields to the real-time access log configuration, the placement of each field in a record can change.

For example, if you add the fields `sc-bytes` and `time-taken` to the real-time access log configuration, these fields are inserted into each log record according to the order shown in the Fields section. The resulting order of all five fields is `time-to-first-byte`, `sc-status`, `sc-bytes`, `time-taken`, and `c-country`. The `c-country` field was originally field number 3, but is now field number 5. Make sure that your consumer application can handle fields that change position in a log record, in case you add fields to your real-time access log configuration.

Troubleshoot real-time access logs

After you create a real-time access log configuration, you might find that no records (or not all records) are delivered to Kinesis Data Streams. In this case, you should first verify that your CloudFront distribution is receiving viewer requests. If it is, you can check the following setting to continue troubleshooting.

IAM role permissions

To deliver real-time access log records to your Kinesis data stream, CloudFront uses the IAM role in the real-time access log configuration. Make sure that the role trust policy and the role permissions policy match the policies shown in IAM role.

Kinesis Data Streams throttling

If CloudFront writes real-time access log records to your Kinesis data stream faster than the stream can handle, Kinesis Data Streams might throttle the requests from CloudFront. In this case, you can increase the number of shards in your Kinesis data stream. Each shard can support writes up to 1,000 records per second, up to a maximum data write of 1 MB per second.

Connection logs

CloudFront connection logs provide detailed visibility into TLS connections, including mutual TLS authentication events, allowing you to monitor certificate validation, track connection attempts, and troubleshoot authentication issues.

What are connection logs?

Connection logs capture detailed information about TLS handshakes and certificate validation for mutual TLS-enabled distributions, and for distribution tenants on the Anycast static IP addresses of their connection group. Unlike standard access logs that record HTTP request information, connection logs focus specifically on the TLS connection establishment phase, including:

- Connection status (success/failure)
- Client certificate details
- TLS protocol and cipher information
- Connection timing metrics
- Custom data from Connection Functions

These logs provide comprehensive visibility into certificate-based authentication events, helping you monitor security, troubleshoot issues, and meet compliance requirements.

Connections that CloudFront logs

CloudFront generates connection logs only for the following connections:

- Connections to a distribution that has mutual TLS (viewer) enabled. For more information, see Mutual TLS authentication with CloudFront (Viewer mTLS).
- HTTPS connections to a distribution tenant on the Anycast static IP addresses of its connection group. For more information, see Connection logs for distribution tenants on Anycast static IPs.

Important

CloudFront doesn't generate connection logs for any other connections. For example, you don't get connection logs for a standard distribution that doesn't use mutual TLS, for distribution tenants on the shared CloudFront IP addresses, or for HTTP connections. You

can enable connection logs on a distribution that doesn't use mutual TLS, but the logs contain only the connections listed above.

To enable connection logs, see Enable connection logs.

Connection logs for distribution tenants on Anycast static IPs

When viewers connect to a distribution tenant through the Anycast static IP addresses of the distribution tenant's connection group, CloudFront can log each HTTPS connection, even when the multi-tenant distribution doesn't use mutual TLS. To get these logs, enable connection logs on the multi-tenant distribution, as described in Enable connection logs. CloudFront doesn't log HTTP connections, or connections to the shared CloudFront IP addresses.

In these records, `distributionId` is the multi-tenant distribution, `distributionTenantId` is the distribution tenant, and `serverIp` is the Anycast static IP address that the viewer connected to. The client certificate fields contain `-`, because the viewer doesn't present a client certificate. The following is an example record:

```
{
    "connectionId": "NSbVU0HLuZtjZEOxeIvLdCmqgCsjq0MRqda1ZxQQVNfCCLrxGZB4-A==",
    "connectionStatus": "Success",
    "clientIp": "192.0.2.110",
    "clientPort": "4996",
    "serverIp": "198.51.100.249",
    "distributionTenantId": "dt_EXAMPLE22222",
    "tlsProtocol": "TLSv1.2",
    "tlsCipher": "ECDHE-RSA-AES128-GCM-SHA256",
    "tlsHandshakeDuration": 70.0,
    "tlsSni": "www.example.com",
    "clientLeafCertSerialNumber": "-",
    "clientLeafCertSubject": "-",
    "clientLeafCertIssuer": "-",
    "clientLeafCertValidity": "-",
    "connectionLogCustomData": "-",
    "eventTimestamp": 1794885872306,
    "distributionId": "EDFDVBD6EXAMPLE"
}
```

Enable connection logs

CloudFront generates connection logs only for connections to distributions with mutual TLS enabled, and for HTTPS connections to distribution tenants on the Anycast static IP addresses of their connection group. For more information, see Connections that CloudFront logs. You can send connection logs to multiple destinations including CloudWatch Logs, Amazon Data Firehose, and Amazon S3.

Prerequisites

Before enabling connection logs:

- Enable mutual TLS for your CloudFront distribution, or serve your distribution tenants from a connection group that uses an Anycast static IP list. CloudFront doesn't generate connection logs for other connections.
- Enable connection logs for your CloudFront distribution
- Make sure you have the required permissions for your chosen logging destination
- For cross-account delivery, configure appropriate IAM policies

To enable connection logs (Console)

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. From the distribution list, select your mTLS-enabled distribution, or the multi-tenant distribution of your distribution tenants on Anycast static IPs.
3. Choose the **Logging** tab.
4. Choose **Add**.
5. Select the service to receive your logs:
   - **CloudWatch Logs**
   - **Firehose**
   - **Amazon S3**
6. For **Destination**, select the resource for your chosen service:
   - For CloudWatch Logs, enter the **Log group name**
   - For Firehose, select the **Firehose delivery stream**

- For Amazon S3, enter the **Bucket name** (optionally with a prefix)

7. (Optional) Configure additional settings:

- **Field selection:** Select specific log fields to include.
- **Output format:** Choose from JSON, Plain, w3c, Raw, or Parquet (S3 only).
- **Field delimiter:** Specify how to separate log fields.

8. Choose **Save changes**

To enable connection logs (AWS CLI)

The following example shows how to enable connection logs using the CloudWatch API:

```
# Step 1: Create a delivery source
aws logs put-delivery-source \
  --name "cf-mtls-connection-logs" \
  --resource-arn "arn:aws:cloudfront::123456789012:distribution/E1A2B3C4D5E6F7" \
  --log-type CONNECTION_LOGS

# Step 2: Create a delivery destination
aws logs put-delivery-destination \
  --name "s3-destination" \
  --delivery-destination-configuration \
  "destinationResourceArn=arn:aws:s3:::amzn-s3-demo-bucket1"

# Step 3: Create the delivery
aws logs create-delivery \
  --delivery-source-name "cf-mtls-connection-logs" \
  --delivery-destination-arn "arn:aws:logs:us-east-1:123456789012:delivery-
destination:s3-destination"
```

> **Note**
>
> When using the CloudWatch API, you must specify the US East (N. Virginia) Region (us-east-1) even when delivering logs to other Regions.

Connection log fields

Connection logs include detailed information about each TLS connection attempt:

Field	Description	Example
eventTimestamp	Unix epoch time in milliseconds when the connection was established or failed	1731620046814
connectionId	Unique identifier for the TLS connection	oLHiEKbQSn8lkvJfA3D4gFowK3_iZ0g4i5nMUjE1Akod8TuAzn5nzg==
connectionStatus	The status of the mTLS connection attempt.	Success or Failed
clientIp	IP address of the connecting client	2001:0db8:85a3:0000:0000:8a2e:0370:7334
clientPort	Port used by the client	12137
serverIp	IP address of the CloudFront edge server	99.84.71.136
distributionId	CloudFront distribution ID	E2DX1SLDPK0123
distributionTenantId	CloudFront distribution tenant ID (when applicable)	dt_2te1Ura9X3R2iCGNjW123
tlsProtocol	TLS protocol version used	TLSv1.3
tlsCipher	TLS cipher suite used for the connection	TLS_AES_128_GCM_SHA256
tlsHandshakeDuration	Duration of the TLS handshake in milliseconds	153
tlsSni	Server Name Indication value from the TLS handshake	d111111abcdef8.cloudfront.net

Field	Description	Example
clientLeafCertSerialNumber	Serial number of the client's certificate	00:b1:43:ed:93:d2:d8:f3:9d
clientLeafCertSubject	Subject field of the client's certificate	C=US, ST=WA, L=Seattle, O=Amazon.com, OU=CloudFront, CN=client.test.mtls.net
clientLeafCertIssuer	Issuer field of the client's certificate	C=US, ST=WA, L=Seattle, O=Amazon.com, OU=CloudFront, CN=test.mtls.net
clientLeafCertValidity	Validity period of the client's certificate	NotBefore=2025-06-05T23:28:21Z;NotAfter=2125-05-12T23:28:21Z
connectionLogCustomData	Custom data added by using Connection Functions	REVOKED:00:b1:43:ed:93:d2:d8:f3:9d

Connection error codes

```
Failed:ClientCertMaxChainDepthExceeded
Failed:ClientCertMaxSizeExceeded
Failed:ClientCertUntrusted
Failed:ClientCertNotYetValid
Failed:ClientCertExpired
Failed:ClientCertTypeUnsupported
Failed:ClientCertInvalid
Failed:ClientCertIntentInvalid
Failed:ClientCertRejected
Failed:ClientCertMissing
Failed:ClientCertOcspProcessingError
Failed:ClientCertOcspDenied
Failed:TcpError
Failed:TcpTimeout
```

```
Failed:ConnectionFunctionError
Failed:ConnectionFunctionDenied
Failed:Internal
Failed:UnmappedConnectionError
```

When connections fail, CloudFront records specific reason codes:

Code	Description
ClientCertMaxChain DepthExceeded	Maximum certificate chain depth exceeded
ClientCertMaxSizeExceeded	Maximum certificate size exceeded
ClientCertUntrusted	Certificate is untrusted
ClientCertNotYetValid	Certificate is not yet valid
ClientCertExpired	Certificate is expired
ClientCertTypeUnsupported	Certificate type is unsupported
ClientCertInvalid	Certificate is invalid
ClientCertIntentInvalid	Certificate intent is invalid
ClientCertRejected	Certificate rejected by custom validation
ClientCertMissing	Certificate is missing
ClientCertOcspProcessingErr or	An error occurred while CloudFront checked the certificate's revocation status with OCSP
ClientCertOcspDenied	The OCSP check didn't confirm that the certificate is valid, for example because the certificate is revoked
TcpError	An error occurred while attempting to establish a connection
TcpTimeout	The connection was not able to be established within the timeout period

Code	Description
ConnectionFunctionError	An uncaught exception was thrown during Connection Function execution
ConnectionFunctionDenied	The Connection Function denied the connection
Internal	An internal service error occurred
UnmappedConnectionError	An error occurred that does not fall into any of the other categories

Edge function logs

You can use Amazon CloudWatch Logs to get logs for your edge functions, both Lambda@Edge and CloudFront Functions. You can access the logs by using the CloudWatch console or the CloudWatch Logs API.

> **Important**
>
> We recommend that you use the logs to understand the nature of the requests for your content, not as a complete accounting of all requests. CloudFront delivers edge function logs on a best-effort basis. The log entry for a particular request might be delivered long after the request was actually processed and, in rare cases, a log entry might not be delivered at all. When a log entry is omitted from edge function logs, the number of entries in the edge function logs won't match the usage that appears in the AWS billing and usage reports.

Topics

- Lambda@Edge logs
- CloudFront Functions logs

Lambda@Edge logs

Lambda@Edge automatically sends function logs to CloudWatch Logs, creating log streams in the AWS Regions where the functions are invoked. When you create or modify a function in AWS Lambda, you can either use the default CloudWatch log group name or customize it.

- The default log group name is `/aws/lambda/<FunctionName>` where `<FunctionName>` is the name that you specified when you created the function. When sending logs to CloudWatch, Lambda@Edge will automatically add the `us-east-1` prefix to the function name, so that the log group name is `/aws/lambda/us-east-1.<FunctionName>`. This prefix corresponds to the AWS Region where the function was created. This prefix remains part of the log group name, even in other Regions where the function is invoked.
- If you specify a custom log group name, such as `/MyLogGroup`, Lambda@Edge won't add the Region prefix. The log group name remains the same across all other Regions where the function is invoked.

> **Note**
>
> If you create a custom log group and specify the same name as the default `/aws/lambda/<FunctionName>`, Lambda@Edge adds the `us-east-1` prefix to the function name.

In addition to customizing the log group name, Lambda@Edge functions support JSON and plain text log formats, and log-level filtering. For more information, see Configuring advanced logging controls for Lambda function in the *AWS Lambda Developer Guide*.

> **Note**
>
> Lambda@Edge throttles logs based on the request volume and the size of logs.

You must review CloudWatch log files in the correct Region to see your Lambda@Edge function log files. To see the Regions where your Lambda@Edge function is running, view graphs of metrics for the function in the CloudFront console. Metrics are displayed for each Region. On the same page, you can choose a Region and then view log files for that Region to investigate issues.

To learn more about how to use CloudWatch Logs with Lambda@Edge functions, see the following topics:

- For more information about viewing graphs in the **Monitoring** section of the CloudFront console, see the section called "Monitor CloudFront metrics with Amazon CloudWatch".
- For information about the permissions required to send data to CloudWatch Logs, see the section called "Set up IAM permissions and roles".
- For information about adding logging to a Lambda@Edge function, see AWS Lambda function logging in Node.js or AWS Lambda function logging in Python in the *AWS Lambda Developer Guide*.
- For information about CloudWatch Logs quotas (formerly known as limits), see CloudWatch Logs quotas in the *Amazon CloudWatch Logs User Guide*.

CloudFront Functions logs

If a CloudFront function's code contains `console.log()` statements, CloudFront Functions automatically sends these log lines to CloudWatch Logs. If there are no `console.log()` statements, nothing is sent to CloudWatch Logs.

CloudFront Functions always creates log streams in the US East (N. Virginia) Region (`us-east-1`), no matter which edge location ran the function. The log stream name is in the format `YYYY/M/D/`*`UUID`*.

The log group name uses the following format:

- For CloudFront Functions at the cache behavior level, the format is `/aws/cloudfront/function/`*`<FunctionName>`*
- For CloudFront Functions at the distribution level (Connection Functions), the format is `/aws/cloudfront/connection-function/`*`<FunctionName>`*

The *`<FunctionName>`* is the name that you gave to the function when you created it. .

Example Viewer requests

The following shows an example log message sent to CloudWatch Logs. Each line begins with an ID that uniquely identifies a CloudFront request. The message begins with a `START` line that includes the CloudFront distribution ID, and ends with an `END` line. Between the `START` and `END` lines are the log lines generated by `console.log()` statements in the function.

```
U7b4hR_RaxMADupvKAvr8_m9gsGXvioUggLV5Oyq-vmAtH8HADpjhw== START DistributionID:
 E3E5D42GADAXZZ
U7b4hR_RaxMADupvKAvr8_m9gsGXvioUggLV5Oyq-vmAtH8HADpjhw== Example function log output
U7b4hR_RaxMADupvKAvr8_m9gsGXvioUggLV5Oyq-vmAtH8HADpjhw== END
```

Example Connection requests

The following shows an example log message sent to CloudWatch Logs. Each line begins with an ID that uniquely identifies a CloudFront request. The message begins with a START line that includes the CloudFront distribution ID, and ends with an END line. Between the START and END lines are the log lines generated by `console.log()` statements in the function.

```
U7b4hR_RaxMADupvKAvr8_m9gsGXvioUggLV5Oyq-vmAtH8HADpjhw== START DistributionID:
 E3E5D42GADA123
U7b4hR_RaxMADupvKAvr8_m9gsGXvioUggLV5Oyq-vmAtH8HADpjhw== 1.2.3.4
U7b4hR_RaxMADupvKAvr8_m9gsGXvioUggLV5Oyq-vmAtH8HADpjhw== END
```

> **Note**
>
> CloudFront Functions sends logs to CloudWatch only for functions in the LIVE stage that run in response to production requests and responses. When you test a function, CloudFront doesn't send any logs to CloudWatch. The test output contains information about errors, compute utilization, and function logs (`console.log()` statements), but this information isn't sent to CloudWatch.

CloudFront Functions uses an AWS Identity and Access Management (IAM) service-linked role to send logs to CloudWatch Logs in your account. A service-linked role is an IAM role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all of the permissions that the service requires to call other AWS services for you. CloudFront Functions uses the **AWSServiceRoleForCloudFrontLogger** service-linked role. For more information about this role, see the section called “Service-linked roles for Lambda@Edge” (Lambda@Edge uses the same service-linked role).

When a function fails with a validation error or an execution error, the information is logged in standard logs and real-time access logs. For specific information about the error, see the `x-edge-result-type`, `x-edge-response-result-type`, and `x-edge-detailed-result-type` fields.

Logging Amazon CloudFront API calls using AWS CloudTrail

CloudFront is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures all API calls for CloudFront as events. The calls captured include calls from the CloudFront console and code calls to the CloudFront API operations. Using the information collected by CloudTrail, you can determine the request that was made to CloudFront, the IP address from which the request was made, when it was made, and additional details.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root user or user credentials.
- Whether the request was made on behalf of an IAM Identity Center user.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

CloudTrail is active in your AWS account when you create the account and you automatically have access to the CloudTrail **Event history**. The CloudTrail **Event history** provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management events in an AWS Region. For more information, see Working with CloudTrail Event history in the *AWS CloudTrail User Guide*. There are no CloudTrail charges for viewing the **Event history**.

For an ongoing record of events in your AWS account past 90 days, create a trail or a CloudTrail Lake event data store.

CloudTrail trails

A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. All trails created using the AWS Management Console are multi-Region. You can create a single-Region or a multi-Region trail by using the AWS CLI. Creating a multi-Region trail is recommended because you capture activity in all AWS Regions in your account. If you create a single-Region trail, you can view only the events logged in the trail's AWS Region. For more information about trails, see Creating a trail for your AWS account and Creating a trail for an organization in the *AWS CloudTrail User Guide*.

You can deliver one copy of your ongoing management events to your Amazon S3 bucket at no charge from CloudTrail by creating a trail, however, there are Amazon S3 storage charges. For

more information about CloudTrail pricing, see AWS CloudTrail Pricing. For information about Amazon S3 pricing, see Amazon S3 Pricing.

CloudTrail Lake event data stores

CloudTrail Lake lets you run SQL-based queries on your events. CloudTrail Lake converts existing events in row-based JSON format to Apache ORC format. ORC is a columnar storage format that is optimized for fast retrieval of data. Events are aggregated into *event data stores*, which are immutable collections of events based on criteria that you select by applying advanced event selectors. The selectors that you apply to an event data store control which events persist and are available for you to query. For more information about CloudTrail Lake, see Working with AWS CloudTrail Lake in the *AWS CloudTrail User Guide*.

CloudTrail Lake event data stores and queries incur costs. When you create an event data store, you choose the pricing option you want to use for the event data store. The pricing option determines the cost for ingesting and storing events, and the default and maximum retention period for the event data store. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

Note

CloudFront is a global service. CloudTrail records events for CloudFront in the US East (N. Virginia) Region. For more information, see Global service events in the *AWS CloudTrail User Guide*.
If you use temporary security credentials by using AWS Security Token Service, calls to regional endpoints, such as `us-west-2`, are logged in CloudTrail to their appropriate Region.
For more information about CloudFront endpoints, see CloudFront endpoints and quotas in the *AWS General Reference*.

CloudFront data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource (for example, reading or writing to a CloudFront distribution). These are also known as data plane operations. Data events are often high-volume activities. By default, CloudTrail doesn't log data events. The CloudTrail **Event history** doesn't record data events.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

You can log data events for the CloudFront resource types by using the CloudTrail console, AWS CLI, or CloudTrail API operations. For more information about how to log data events, see Logging data events with the AWS Management Console and Logging data events with the AWS Command Line Interface in the *AWS CloudTrail User Guide*.

The following table lists the CloudFront resource types for which you can log data events. The **Data event type (console)** column shows the value to choose from the **Data event type** list on the CloudTrail console. The **resources.type value** column shows the `resources.type` value, which you would specify when configuring advanced event selectors using the AWS CLI or CloudTrail APIs. The **Data APIs logged to CloudTrail** column shows the API calls logged to CloudTrail for the resource type.

Data event type (console)	resources.type value	Data APIs logged to CloudTrail
CloudFront KeyValueStore	`AWS::CloudFront::KeyValueStore`	• DeleteKeys • DescribeKeyValueStore • GetKey • ListKeys • PutKeys • UpdateKeys

You can configure advanced event selectors to filter on the `eventName`, `readOnly`, and `resources.ARN` fields to log only those events that are important to you. For more information about these fields, see AdvancedFieldSelector in the *AWS CloudTrail API Reference*.

CloudFront management events in CloudTrail

Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations. By default, CloudTrail logs management events.

Amazon CloudFront logs all CloudFront control plane operations as management events. For a list of the Amazon CloudFront control plane operations that CloudFront logs to CloudTrail, see the Amazon CloudFront API Reference.

CloudFront event examples

An event represents a single request from any source and includes information about the requested API operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so events don't appear in any specific order.

Contents

- Example: UpdateDistribution
- Example: UpdateKeys

Example: UpdateDistribution

The following example shows a CloudTrail event that demonstrates the UpdateDistribution operation.

For calls to the CloudFront API, the `eventSource` is `cloudfront.amazonaws.com`.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:role-session-name",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/role-session-name",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-02-02T19:23:50Z",
```

```
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-02-02T19:26:01Z",
    "eventSource": "cloudfront.amazonaws.com",
    "eventName": "UpdateDistribution",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "52.94.133.137",
    "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/121.0.0.0 Safari/537.36",
    "requestParameters": {
        "distributionConfig": {
            "defaultRootObject": "",
            "aliases": {
                "quantity": 3,
                "items": [
                    "alejandro_rosalez.awsps.myinstance.com",
                    "cross-testing.alejandro_rosalez.awsps.myinstance.com",
                    "*.alejandro_rosalez.awsps.myinstance.com"
                ]
            },
            "cacheBehaviors": {
                "quantity": 0,
                "items": []
            },
            "httpVersion": "http2and3",
            "originGroups": {
                "quantity": 0,
                "items": []
            },
            "viewerCertificate": {
                "minimumProtocolVersion": "TLSv1.2_2021",
                "cloudFrontDefaultCertificate": false,
                "aCMCertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                "sSLSupportMethod": "sni-only"
            },
            "webACLId": "arn:aws:wafv2:us-east-1:111122223333:global/webacl/testing-
acl/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "customErrorResponses": {
                "quantity": 0,
                "items": []
            },
```

```
            "logging": {
                "includeCookies": false,
                "prefix": "",
                "enabled": false,
                "bucket": ""
            },
            "priceClass": "PriceClass_All",
            "restrictions": {
                "geoRestriction": {
                    "restrictionType": "none",
                    "quantity": 0,
                    "items": []
                }
            },
            "isIPV6Enabled": true,
            "callerReference": "1578329170895",
            "continuousDeploymentPolicyId": "",
            "enabled": true,
            "defaultCacheBehavior": {
                "targetOriginId": "d111111abcdef8",
                "minTTL": 0,
                "compress": false,
                "maxTTL": 31536000,
                "functionAssociations": {
                    "quantity": 0,
                    "items": []
                },
                "trustedKeyGroups": {
                    "quantity": 0,
                    "items": [],
                    "enabled": false
                },
                "smoothStreaming": false,
                "fieldLevelEncryptionId": "",
                "defaultTTL": 86400,
                "lambdaFunctionAssociations": {
                    "quantity": 0,
                    "items": []
                },
                "viewerProtocolPolicy": "redirect-to-https",
                "forwardedValues": {
                    "cookies": {"forward": "none"},
                    "queryStringCacheKeys": {
                        "quantity": 0,
```

```
                    "items": []
                },
                "queryString": false,
                "headers": {
                    "quantity": 1,
                    "items": ["*"]
                }
            },
            "trustedSigners": {
                "items": [],
                "enabled": false,
                "quantity": 0
            },
            "allowedMethods": {
                "quantity": 2,
                "items": [
                    "HEAD",
                    "GET"
                ],
                "cachedMethods": {
                    "quantity": 2,
                    "items": [
                        "HEAD",
                        "GET"
                    ]
                }
            }
        },
        "staging": false,
        "origins": {
            "quantity": 1,
            "items": [
                {
                    "originPath": "",
                    "connectionTimeout": 10,
                    "customOriginConfig": {
                        "originReadTimeout": 30,
                        "hTTPSPort": 443,
                        "originProtocolPolicy": "https-only",
                        "originKeepaliveTimeout": 5,
                        "hTTPPort": 80,
                        "originSslProtocols": {
                            "quantity": 3,
                            "items": [
```

```
                                "TLSv1",
                                "TLSv1.1",
                                "TLSv1.2"
                            ]
                        }
                    },
                    "id": "d111111abcdef8",
                    "domainName": "d111111abcdef8.cloudfront.net",
                    "connectionAttempts": 3,
                    "customHeaders": {
                        "quantity": 0,
                        "items": []
                    },
                    "originShield": {"enabled": false},
                    "originAccessControlId": ""
                }
            ]
        },
        "comment": "HIDDEN_DUE_TO_SECURITY_REASONS"
    },
    "id": "EDFDVBD6EXAMPLE",
    "ifMatch": "E1RTLUR9YES76O"
},
"responseElements": {
    "distribution": {
        "activeTrustedSigners": {
            "quantity": 0,
            "enabled": false
        },
        "id": "EDFDVBD6EXAMPLE",
        "domainName": "d111111abcdef8.cloudfront.net",
        "distributionConfig": {
            "defaultRootObject": "",
            "aliases": {
                "quantity": 3,
                "items": [
                    "alejandro_rosalez.awsps.myinstance.com",
                    "cross-testing.alejandro_rosalez.awsps.myinstance.com",
                    "*.alejandro_rosalez.awsps.myinstance.com"
                ]
            },
            "cacheBehaviors": {"quantity": 0},
            "httpVersion": "http2and3",
            "originGroups": {"quantity": 0},
```

```
                "viewerCertificate": {
                    "minimumProtocolVersion": "TLSv1.2_2021",
                    "cloudFrontDefaultCertificate": false,
                    "aCMCertificateArn": "arn:aws:acm:us-
east-1:111122223333:certificate/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
                    "sSLSupportMethod": "sni-only",
                    "certificateSource": "acm",
                    "certificate": "arn:aws:acm:us-east-1:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                },
                "webACLId": "arn:aws:wafv2:us-east-1:111122223333:global/webacl/
testing-acl/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
                "customErrorResponses": {"quantity": 0},
                "logging": {
                    "includeCookies": false,
                    "prefix": "",
                    "enabled": false,
                    "bucket": ""
                },
                "priceClass": "PriceClass_All",
                "restrictions": {
                    "geoRestriction": {
                        "restrictionType": "none",
                        "quantity": 0
                    }
                },
                "isIPV6Enabled": true,
                "callerReference": "1578329170895",
                "continuousDeploymentPolicyId": "",
                "enabled": true,
                "defaultCacheBehavior": {
                    "targetOriginId": "d111111abcdef8",
                    "minTTL": 0,
                    "compress": false,
                    "maxTTL": 31536000,
                    "functionAssociations": {"quantity": 0},
                    "trustedKeyGroups": {
                        "quantity": 0,
                        "enabled": false
                    },
                    "smoothStreaming": false,
                    "fieldLevelEncryptionId": "",
                    "defaultTTL": 86400,
                    "lambdaFunctionAssociations": {"quantity": 0},
```

```
                    "viewerProtocolPolicy": "redirect-to-https",
                    "forwardedValues": {
                        "cookies": {"forward": "none"},
                        "queryStringCacheKeys": {"quantity": 0},
                        "queryString": false,
                        "headers": {
                            "quantity": 1,
                            "items": ["*"]
                        }
                    },
                    "trustedSigners": {
                        "enabled": false,
                        "quantity": 0
                    },
                    "allowedMethods": {
                        "quantity": 2,
                        "items": [
                            "HEAD",
                            "GET"
                        ],
                        "cachedMethods": {
                            "quantity": 2,
                            "items": [
                                "HEAD",
                                "GET"
                            ]
                        }
                    }
                },
                "staging": false,
                "origins": {
                    "quantity": 1,
                    "items": [
                        {
                            "originPath": "",
                            "connectionTimeout": 10,
                            "customOriginConfig": {
                                "originReadTimeout": 30,
                                "hTTPSPort": 443,
                                "originProtocolPolicy": "https-only",
                                "originKeepaliveTimeout": 5,
                                "hTTPPort": 80,
                                "originSslProtocols": {
                                    "quantity": 3,
```

```
                                "items": [
                                    "TLSv1",
                                    "TLSv1.1",
                                    "TLSv1.2"
                                ]
                            }
                        },
                        "id": "d111111abcdef8",
                        "domainName": "d111111abcdef8.cloudfront.net",
                        "connectionAttempts": 3,
                        "customHeaders": {"quantity": 0},
                        "originShield": {"enabled": false},
                        "originAccessControlId": ""
                    }
                ]
            },
            "comment": "HIDDEN_DUE_TO_SECURITY_REASONS"
        },
        "aliasICPRecordals": [
            {
                "cNAME": "alejandro_rosalez.awsps.myinstance.com",
                "iCPRecordalStatus": "APPROVED"
            },
            {
                "cNAME": "cross-testing.alejandro_rosalez.awsps.myinstance.com",
                "iCPRecordalStatus": "APPROVED"
            },
            {
                "cNAME": "*.alejandro_rosalez.awsps.myinstance.com",
                "iCPRecordalStatus": "APPROVED"
            }
        ],
        "aRN": "arn:aws:cloudfront::111122223333:distribution/EDFDVBD6EXAMPLE",
        "status": "InProgress",
        "lastModifiedTime": "Feb 2, 2024 7:26:01 PM",
        "activeTrustedKeyGroups": {
            "enabled": false,
            "quantity": 0
        },
        "inProgressInvalidationBatches": 0
    },
    "eTag": "E1YHBLAB2BJY1G"
},
"requestID": "4e6b66f9-d548-11e3-a8a9-73e33example",
```

```
    "eventID": "5ab02562-0fc5-43d0-b7b6-90293example",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "apiVersion": "2020_05_31",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "cloudfront.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

Example: UpdateKeys

The following example shows a CloudTrail event that demonstrates the UpdateKeys operation.

For calls to the CloudFront KeyValueStore API, the `eventSource` is `edgekeyvaluestore.amazonaws.com` instead of `cloudfront.amazonaws.com`.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:role-session-name",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/role-session-name",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2023-11-01T23:41:14Z",
                "mfaAuthenticated": "false"
            }
        }
```

```
    },
    "eventTime": "2023-11-01T23:41:28Z",
    "eventSource": "edgekeyvaluestore.amazonaws.com",
    "eventName": "UpdateKeys",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "3.235.183.252",
    "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/121.0.0.0 Safari/537.36",
    "requestParameters": {
        "kvsARN": "arn:aws:cloudfront::111122223333:key-value-store/a1b2c3d4-5678-90ab-
cdef-EXAMPLE11111",
        "ifMatch": "KV3O6B1CX531EBP",
        "deletes": [
            {"key": "key1"}
        ]
    },
    "responseElements": {
        "itemCount": 0,
        "totalSizeInBytes": 0,
        "eTag": "KVDC9VEVZ71ZGO"
    },
    "requestID": "5ccf104c-acce-4ea1-b7fc-73e33example",
    "eventID": "a0b1b5c7-906c-439d-9925-90293example",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::CloudFront::KeyValueStore",
            "ARN": "arn:aws:cloudfront::111122223333:key-value-store/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "111122223333.cloudfront-kvs.global.api.aws"
    }
}
```

For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Track configuration changes with AWS Config

To record and evaluate configurations of your AWS resources, you can use AWS Config, which provides you with a detailed view of the configuration of your distributions. This includes how the resources are related to one another and how they were configured in the past, so you can review changes over time.

You can also use AWS Config to record configuration changes to your CloudFront distribution settings. You can capture changes to distribution states, price classes, origins, geographic restriction settings, and Lambda@Edge configurations.

> **Note**
>
> AWS Config does not record key–value tags for CloudFront streaming distributions.

Contents

- Set up AWS Config with CloudFront
- View CloudFront configuration history
- Evaluate CloudFront configurations with AWS Config Rules

Set up AWS Config with CloudFront

When you set up AWS Config, you can choose to record all supported AWS resources or record only some specified resources, such as recording changes for CloudFront only. For a list of supported CloudFront resources, see the Amazon CloudFront section of the Supported Resource Types topic in the *AWS Config Developer Guide*.

> **Notes**
>
> - To track configuration changes to your CloudFront distribution, you must sign in to the CloudFront console in the US East (N. Virginia) AWS Region.

- There might be a delay in recording resources with AWS Config. AWS Config records resources only after it discovers the resources.

Console

To set up AWS Config with CloudFront

1. Sign in to the AWS Management Console and open the AWS Config console.
2. Choose **Get Started Now**.
3. On the **Settings** page, for **Resource types to record**, specify the AWS resource types that you want AWS Config to record. If you want to record only CloudFront changes, choose **Specific types**, and then, under **CloudFront**, choose the distribution or streaming distribution that you want to track changes for.

   To add or change which distributions to track, choose **Settings** on the left, after completing your initial setup.
4. Specify additional required options for AWS Config: set up a notification, specify a location for the configuration information, and add rules for evaluating resource types.

For more information, see Setting up AWS Config with the Console in the *AWS Config Developer Guide*.

AWS CLI

To set up AWS Config with CloudFront using the AWS CLI, see Setting up AWS Config with the AWS CLI in the *AWS Config Developer Guide*.

AWS Config API

To set up AWS Config with CloudFront using the AWS Config API, see the StartConfigurationRecorder API operation in the *AWS Config API Reference*.

View CloudFront configuration history

After AWS Config starts recording configuration changes to your distributions, you can get the configuration history of any distribution that you have configured for CloudFront.

You can view configuration histories in the following ways.

Console

For each recorded resource, you can view a timeline page that provides a history of configuration details. To view this page, choose the gray icon in the **Config Timeline** column of the **Dedicated Hosts** page.

For more information, see Viewing Configuration Details in the AWS Config Console in the *AWS Config Developer Guide*.

AWS CLI

To get a list of all your distributions, run the list-discovered-resources command, as shown in the following example.

```
aws configservice list-discovered-resources --resource-type
 AWS::CloudFront::Distribution
```

To get the configuration details of a distribution for a specific time interval, run the get-resource-config-history command.

For more information, see View Configuration Details Using the CLI in the *AWS Config Developer Guide*.

AWS Config API

To get a list of all your distributions, use the ListDiscoveredResources API operation.

To get the configuration details of a distribution for a specific time interval, use the GetResourceConfigHistory API operation. For more information, see the AWS Config API Reference.

Evaluate CloudFront configurations with AWS Config Rules

You can evaluate configurations against desired configurations with AWS Config Rules. For example, AWS Config Rules helps you to evaluate whether your CloudFront resources comply with common security best practices. You can choose managed rules like viewer policy HTTPS, SNI enabled, OAC enabled, origin failover enabled, AWS WAF WebACL, or AWS Shield Advanced resource policies to be triggered when the configuration changes.

Managed rules can run evaluations periodically, at a frequency that you choose. AWS Firewall Manager relies on AWS Config for automatic alerts and remediations. For more information, see

Evaluating Resources with AWS Config Rules and List of AWS Config Managed Rules in the *AWS Config Developer Guide*.

Security in Amazon CloudFront

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon CloudFront, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using CloudFront. The following topics show you how to configure CloudFront to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CloudFront resources.

- Data protection in Amazon CloudFront – Protect your data in transit and at rest.
- Identity and Access Management for Amazon CloudFront – Manage access to CloudFront resources and permissions.
- Logging and monitoring in Amazon CloudFront – Track requests and monitor the health of your distributions.
- Compliance validation for Amazon CloudFront – Understand how CloudFront meets compliance programs.
- Resilience in Amazon CloudFront – Learn about the resilience of the CloudFront infrastructure.
- Infrastructure security in Amazon CloudFront – Learn about infrastructure security for CloudFront.

Topics

- Data protection in Amazon CloudFront
- Identity and Access Management for Amazon CloudFront
- Logging and monitoring in Amazon CloudFront
- Compliance validation for Amazon CloudFront
- Resilience in Amazon CloudFront
- Infrastructure security in Amazon CloudFront

Data protection in Amazon CloudFront

The AWS shared responsibility model applies to data protection in Amazon CloudFront. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with CloudFront or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Amazon CloudFront provides several options that you can use to help secure the content that it delivers:

- Configure HTTPS connections.
- Configure field-level encryption to provide additional security for specific data during transit.
- Restrict access to content so that only specific people, or people in a specific area, can view it.

The following topics explain the options in more detail.

Topics

- Encryption in transit
- Encryption at rest
- Restrict access to content

Encryption in transit

To encrypt your data during transit, you configure Amazon CloudFront to require that viewers use HTTPS to request your files, so that connections are encrypted when CloudFront communicates with viewers. You also can configure CloudFront to use HTTPS to get files from your origin, so that connections are encrypted when CloudFront communicates with your origin.

For more information, see Use HTTPS with CloudFront.

Field-level encryption adds an additional layer of security along with HTTPS that lets you protect specific data throughout system processing so that only certain applications can see it. By configuring field-level encryption in CloudFront, you can securely upload user-submitted sensitive information to your web servers. The sensitive information provided by your clients is encrypted at the edge closer to the user. It remains encrypted throughout your entire application stack, making sure that only applications that need the data—and have the credentials to decrypt it—are able to do so.

For more information, see Use field-level encryption to help protect sensitive data.

The CloudFront API endpoints, `cloudfront.amazonaws.com` and `cloudfront-fips.amazonaws.com`, only accept HTTPS traffic. This means that when you send and receive information using the CloudFront API, your data is always encrypted in transit. This data includes distribution configurations, cache policies and origin request policies, key groups and public keys, and function code in CloudFront Functions. In addition, all requests sent to the CloudFront API endpoints are signed with AWS credentials and logged in AWS CloudTrail.

Function code and configuration in CloudFront Functions is always encrypted in transit when copied to the edge location points of presence (POPs), and between other storage locations used by CloudFront.

Encryption at rest

Function code and configuration in CloudFront Functions is always stored in an encrypted format on the edge location POPs, and in other storage locations used by CloudFront.

Restrict access to content

Many companies that distribute content over the internet want to restrict access to documents, business data, media streams, or content that is intended for a subset of users. To securely serve this content by using Amazon CloudFront, you can do one or more of the following:

Use signed URLs or cookies

You can restrict access to content that is intended for selected users—for example, users who have paid a fee—by serving this private content through CloudFront using signed URLs or signed cookies. For more information, see Serve private content with signed URLs and signed cookies.

Restrict access to content in Amazon S3 buckets

If you restrict access to your content by using, for example, CloudFront signed URLs or signed cookies, you also won't want people to view files by using the direct URL for the file. Instead, you want them to access the files only by using the CloudFront URL, so that your protections work.

If you use an Amazon S3 bucket as the origin for a CloudFront distribution, you can set up an origin access control (OAC) which makes it possible to restrict access to the S3 bucket. For more information, see the section called “Restrict access to an Amazon S3 origin”.

Restrict access to content served by an Application Load Balancer

When you use CloudFront with an Application Load Balancer in Elastic Load Balancing as the origin, you can configure CloudFront to prevent users from directly accessing the Application Load Balancer. This allows users to access the Application Load Balancer only through CloudFront, making sure that you get the benefits of using CloudFront. For more information, see Restrict access to Application Load Balancers.

Use AWS WAF web ACLs

You can use AWS WAF, a web application firewall service, to create a web access control list (web ACL) to restrict access to your content. Based on conditions that you specify, such as the IP addresses that requests originate from or the values of query strings, CloudFront responds to requests either with the requested content or with an HTTP 403 status code (Forbidden). For more information, see Use AWS WAF protections.

Use geo restriction

You can use *geo restriction*, also known as *geoblocking*, to prevent users in specific geographic locations from accessing content that you serve through a CloudFront distribution. There are several options to choose from when you configure geo restrictions. For more information, see Restrict the geographic distribution of your content.

Identity and Access Management for Amazon CloudFront

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use CloudFront resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon CloudFront works with IAM
- Identity-based policy examples for Amazon CloudFront
- AWS managed policies for Amazon CloudFront

- Use service-linked roles for CloudFront
- Troubleshoot Amazon CloudFront identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshoot Amazon CloudFront identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon CloudFront works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for Amazon CloudFront)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon CloudFront works with IAM

Before you use IAM to manage access to CloudFront, learn what IAM features are available to use with CloudFront.

IAM features you can use with Amazon CloudFront

IAM feature	CloudFront support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Forward access sessions (FAS)	No
Service roles	No
Service-linked roles	Yes

To get a high-level view of how CloudFront and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for CloudFront

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for CloudFront

To view examples of CloudFront identity-based policies, see Identity-based policy examples for Amazon CloudFront.

Resource-based policies within CloudFront

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for CloudFront

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of CloudFront actions, see Actions defined by Amazon CloudFront in the *Service Authorization Reference*.

Policy actions in CloudFront use the following prefix before the action:

```
cloudfront
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "cloudfront:action1",
      "cloudfront:action2"
         ]
```

To view examples of CloudFront identity-based policies, see Identity-based policy examples for Amazon CloudFront.

Policy resources for CloudFront

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of CloudFront resource types and their ARNs, see Resources defined by Amazon CloudFront in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon CloudFront.

To view examples of CloudFront identity-based policies, see Identity-based policy examples for Amazon CloudFront.

Policy condition keys for CloudFront

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of CloudFront condition keys, see Condition keys for Amazon CloudFront in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon CloudFront.

To view examples of CloudFront identity-based policies, see Identity-based policy examples for Amazon CloudFront.

ACLs in CloudFront

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with CloudFront

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

CloudFront supports ABAC for CloudFront resources that support tags, such as distributions, tenants, or trust stores.

Using temporary credentials with CloudFront

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Forward access sessions for CloudFront

Supports forward access sessions (FAS): No

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for CloudFront

Supports service roles: No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break CloudFront functionality. Edit service roles only when CloudFront provides guidance to do so.

Service-linked roles for CloudFront

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

CloudFront uses service-linked roles to perform actions for you. For more information about creating or managing CloudFront service-linked roles, see Use service-linked roles for CloudFront. For more information about creating or managing Lambda@Edge service-linked roles, see Service-linked roles for Lambda@Edge.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon CloudFront

By default, users and roles don't have permission to create or modify CloudFront resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by CloudFront, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon CloudFront in the *Service Authorization Reference*.

Topics

- Policy best practices
- Allow users to view their own permissions
- Permissions to access CloudFront programmatically
- Permissions required to use the CloudFront console
- Customer managed policy examples

Policy best practices

Identity-based policies determine whether someone can create, access, or delete CloudFront resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws-eusc:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Permissions to access CloudFront programmatically

The following shows a permissions policy. The `Sid`, or statement ID, is optional.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowAllCloudFrontPermissions",
         "Effect": "Allow",
         "Action": ["cloudfront:*"],
         "Resource": "*"
      }
   ]
}
```

The policy grants permissions to perform all CloudFront operations, which is sufficient to access CloudFront programmatically. If you're using the console to access CloudFront, see Permissions required to use the CloudFront console.

For a list of actions and the ARN that you specify to grant or deny permission to use each action, see Actions, resources, and condition keys for Amazon CloudFront in the *Service Authorization Reference*.

Permissions required to use the CloudFront console

To grant full access to the CloudFront console, you grant the permissions in the following permissions policy:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm:ListCertificates",
            "cloudfront:*",
            "cloudwatch:DescribeAlarms",
            "cloudwatch:PutMetricAlarm",
            "cloudwatch:GetMetricStatistics",
```

```
            "elasticloadbalancing:DescribeLoadBalancers",
            "iam:ListServerCertificates",
            "sns:ListSubscriptionsByTopic",
            "sns:ListTopics",
            "waf:GetWebACL",
            "waf:ListWebACLs"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets",
            "s3:PutBucketPolicy"
         ],
         "Resource":"arn:aws-eusc:s3:::*"
      }
   ]
}
```

Here's why the permissions are required:

`acm:ListCertificates`

When you're creating and updating distributions by using the CloudFront console and you want to configure CloudFront to require HTTPS between the viewer and CloudFront or between CloudFront and the origin, you can view a list of ACM certificates.

This permission isn't required if you aren't using the CloudFront console.

`cloudfront:*`

You can perform all CloudFront actions.

`cloudwatch:DescribeAlarms` and `cloudwatch:PutMetricAlarm`

Let you create and view CloudWatch alarms in the CloudFront console. See also `sns:ListSubscriptionsByTopic` and `sns:ListTopics`.

These permissions aren't required if you aren't using the CloudFront console.

`cloudwatch:GetMetricStatistics`

Lets CloudFront render CloudWatch metrics in the CloudFront console.

This permission isn't required if you aren't using the CloudFront console.

`elasticloadbalancing:DescribeLoadBalancers`

When creating and updating distributions, you can view a list of Elastic Load Balancing load balancers in the list of available origins.

This permission isn't required if you aren't using the CloudFront console.

`iam:ListServerCertificates`

When you're creating and updating distributions by using the CloudFront console and you want to configure CloudFront to require HTTPS between the viewer and CloudFront or between CloudFront and the origin, you can view a list of certificates in the IAM certificate store.

This permission isn't required if you aren't using the CloudFront console.

`s3:ListAllMyBuckets`

When you're creating and updating distributions, you can perform the following operations:

- View a list of S3 buckets in the list of available origins
- View a list of S3 buckets that you can save access logs in

This permission isn't required if you aren't using the CloudFront console.

`s3:PutBucketPolicy`

When you're creating or updating distributions that restrict access to S3 buckets, lets a user update the bucket policy to grant access to the CloudFront origin access identity. For more information, see the section called “Use an origin access identity (legacy, not recommended)”.

This permission isn't required if you aren't using the CloudFront console.

`sns:ListSubscriptionsByTopic` and `sns:ListTopics`

When you create CloudWatch alarms in the CloudFront console, you can choose an SNS topic for notifications.

These permissions aren't required if you aren't using the CloudFront console.

`waf:GetWebACL` and `waf:ListWebACLs`

You can view a list of AWS WAF web ACLs in the CloudFront console.

These permissions aren't required if you aren't using the CloudFront console.

Permission-only actions for the CloudFront console

You can perform the following CloudFront actions on the CloudFront Security Savings Bundle page. The following API actions are not intended to be called by your code, and are not included in the AWS CLI and AWS SDKs.

Action	Description
`CreateSavingsPlan`	Grants permission to create a new savings plan.
`GetSavingsPlan`	Grants permission to get a savings plan.
`ListRateCards`	Grants permission to list CloudFront rate cards for the account.
`ListSavingsPlans`	Grants permission to list savings plans in the account.
`ListUsages`	Grants permission to list CloudFront usage.
`UpdateSavingsPlan`	Grants permission to update a savings plan.

Notes

- For more information about CloudFront savings plans, see the CloudFront Security Savings Bundle section of the Amazon CloudFront FAQs.
- If you create a savings plan for CloudFront and then want to delete it later, contact AWS Support.

Customer managed policy examples

You can create your own custom IAM policies to allow permissions for CloudFront API actions. You can attach these custom policies to the IAM users or groups that require the specified permissions.

These policies work when you are using the CloudFront API, the AWS SDKs, or the AWS CLI. The following examples show permissions for a few common use cases. For the policy that grants a user full access to CloudFront, see Permissions required to use the CloudFront console.

Examples

- Example 1: Allow read access to all distributions
- Example 2: Allow creating, updating, and deleting distributions
- Example 3: Allow creating and listing invalidations
- Example 4: Allow creating a distribution

Example 1: Allow read access to all distributions

The following permissions policy grants the user permissions to view all distributions in the CloudFront console:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm:ListCertificates",
            "cloudfront:GetDistribution",
            "cloudfront:GetDistributionConfig",
            "cloudfront:ListDistributions",
            "cloudfront:ListCloudFrontOriginAccessIdentities",
            "elasticloadbalancing:DescribeLoadBalancers",
            "iam:ListServerCertificates",
            "sns:ListSubscriptionsByTopic",
            "sns:ListTopics",
            "waf:GetWebACL",
            "waf:ListWebACLs"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
```

```
            "s3:ListAllMyBuckets"
         ],
         "Resource":"arn:aws-eusc:s3:::*"
      }
   ]
}
```

Example 2: Allow creating, updating, and deleting distributions

With the following permissions policy, you can create, update, and delete distributions by using the CloudFront console:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm:ListCertificates",
            "cloudfront:CreateDistribution",
            "cloudfront:DeleteDistribution",
            "cloudfront:GetDistribution",
            "cloudfront:GetDistributionConfig",
            "cloudfront:ListDistributions",
            "cloudfront:UpdateDistribution",
            "cloudfront:ListCloudFrontOriginAccessIdentities",
            "elasticloadbalancing:DescribeLoadBalancers",
            "iam:ListServerCertificates",
            "sns:ListSubscriptionsByTopic",
            "sns:ListTopics",
            "waf:GetWebACL",
            "waf:ListWebACLs"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets",
            "s3:PutBucketPolicy"
```

```
        ],
        "Resource":"arn:aws-eusc:s3:::*"
      }
   ]
}
```

The `cloudfront:ListCloudFrontOriginAccessIdentities` permission allows users to automatically grant to an existing origin access identity the permission to access objects in an Amazon S3 bucket. If you also want users to be able to create origin access identities, you also need to allow the `cloudfront:CreateCloudFrontOriginAccessIdentity` permission.

Example 3: Allow creating and listing invalidations

With the following permissions policy, you can create and list invalidations. It includes read access to CloudFront distributions because you create and view invalidations by first displaying settings for a distribution:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm:ListCertificates",
            "cloudfront:GetDistribution",
            "cloudfront:GetStreamingDistribution",
            "cloudfront:GetDistributionConfig",
            "cloudfront:ListDistributions",
            "cloudfront:ListCloudFrontOriginAccessIdentities",
            "cloudfront:CreateInvalidation",
            "cloudfront:GetInvalidation",
            "cloudfront:ListInvalidations",
            "elasticloadbalancing:DescribeLoadBalancers",
            "iam:ListServerCertificates",
            "sns:ListSubscriptionsByTopic",
            "sns:ListTopics",
            "waf:GetWebACL",
            "waf:ListWebACLs"
         ],
```

```
            "Resource":"*"
        },
        {
            "Effect":"Allow",
            "Action":[
                "s3:ListAllMyBuckets"
            ],
            "Resource":"arn:aws-eusc:s3:::*"
        }
    ]
}
```

Example 4: Allow creating a distribution

The following permission policy grants the user permission to create and list distributions in the CloudFront console. For the `CreateDistribution` action, specify the wildcard (*) character for the `Resource` instead of a wildcard for the distribution ARN (`arn:aws:cloudfront::123456789012:distribution/*`). For more information about the `Resource` element, see IAM JSON policy elements: Resource in the *IAM User Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "cloudfront:CreateDistribution",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "cloudfront:ListDistributions",
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for Amazon CloudFront

To add permissions to users, groups, and roles, it's easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your users with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new permissions become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

Topics

- AWS managed policy: CloudFrontReadOnlyAccess
- AWS managed policy: CloudFrontFullAccess
- AWS managed policy: AWSCloudFrontLogger
- AWS managed policy: AWSLambdaReplicator
- AWS managed policy: AWSCloudFrontVPCOriginServiceRolePolicy
- CloudFront updates to AWS managed policies

AWS managed policy: CloudFrontReadOnlyAccess

You can attach the **CloudFrontReadOnlyAccess** policy to your IAM identities. This policy allows read-only permissions to CloudFront resources. It also allows read-only permissions to other AWS service resources that are related to CloudFront and that are visible in the CloudFront console.

Permissions details

This policy includes the following permissions.

- `cloudfront:Describe*` – Allows principals to get information about metadata about CloudFront resources.
- `cloudfront:Get*` – Allows principals to get detailed information and configurations for CloudFront resources.
- `cloudfront:List*` – Allows principals to get lists of CloudFront resources.
- `cloudfront-keyvaluestore:Describe*` - Allows principals to get information about the key value store.
- `cloudfront-keyvaluestore:Get*` - Allows principals to get detailed information and configurations for the key value store.
- `cloudfront-keyvaluestore:List*` - Allows principals to get lists of the key value stores.
- `acm:DescribeCertificate` – Allows principals to get details about an ACM certificate.
- `acm:ListCertificates` – Allows principals to get a list of ACM certificates.
- `iam:ListServerCertificates` – Allows principals to get a list of server certificates stored in IAM.
- `route53:List*` – Allows principals to get lists of Route 53 resources.
- `waf:ListWebACLs` – Allows principals to get a list of web ACLs in AWS WAF.
- `waf:GetWebACL` – Allows principals to get detailed information about web ACLs in AWS WAF.
- `wafv2:ListWebACLs` – Allows principals to get a list of web ACLs in AWS WAF.
- `wafv2:GetWebACL` – Allows principals to get detailed information about web ACLs in AWS WAF.
- `pricingplanmanager:GetSubscription` – Allows principals read-only access to get details about pricing plan subscriptions.
- `pricingplanmanager:ListSubscriptions` – Allows principals read-only access to list pricing plan subscriptions.
- `ec2:DescribeIpamPools` – Allows principals to get detailed information about your IPAM pools.

- `ec2:GetIpamPoolCidrs` – Allows principals to get the CIDRs provisioned to an IPAM pool.

To view the permissions for this policy, see CloudFrontReadOnlyAccess in the *AWS Managed Policy Reference*.

AWS managed policy: CloudFrontFullAccess

You can attach the **CloudFrontFullAccess** policy to your IAM identities. This policy allows administrative permissions to CloudFront resources. It also allows read-only permissions to other AWS service resources that are related to CloudFront and that are visible in the CloudFront console.

Permissions details

This policy includes the following permissions.

- `s3:ListAllMyBuckets` – Allows principals to get a list of all Amazon S3 buckets.
- `acm:DescribeCertificate` – Allows principals to get details about an ACM certificate.
- `acm:ListCertificates` – Allows principals to get a list of ACM certificates.
- `acm:RequestCertificate` – Allows principals to request managed certificates from ACM.
- `cloudfront:*` – Allows principals to perform all actions on all CloudFront resources.
- `cloudfront-keyvaluestore:*` - Allows principals to perform all actions on the key value store.
- `iam:ListServerCertificates` – Allows principals to get a list of server certificates stored in IAM.
- `waf:ListWebACLs` – Allows principals to get a list of web ACLs in AWS WAF.
- `waf:GetWebACL` – Allows principals to get detailed information about web ACLs in AWS WAF.
- `waf:CreateWebACLs` – Allows principals to create a web ACL in AWS WAF.
- `wafv2:ListWebACLs` – Allows principals to get a list of web ACLs in AWS WAF.
- `wafv2:GetWebACL` – Allows principals to get detailed information about web ACLs in AWS WAF.
- `kinesis:ListStreams` – Allows principals to get a list of Amazon Kinesis streams.
- `elasticloadbalancing:DescribeLoadBalancers` - Allows principals to get detailed information about load balancers in Elastic Load Balancing.
- `kinesis:DescribeStream` – Allows principals to get detailed information about a Kinesis stream.

- `iam:ListRoles` – Allows principals to get a list of roles in IAM.
- `pricingplanmanager:AssociateResourcesToSubscription` - Allows principals to associate resources to a subscription. This enables the resources to be covered by the subscription's pricing plan.
- `pricingplanmanager:CancelSubscription` - Allows principals to cancel an existing subscription.
- `pricingplanmanager:CancelSubscriptionChange` - Allows principals to cancel a pending change to an existing subscription, such as a plan upgrade, before the change is applied.
- `pricingplanmanager:CreateSubscription` - Allows principals to create a subscription to a pricing plan.
- `pricingplanmanager:DisassociateResourcesFromSubscription` - Allows principals to remove the association between resources and an existing subscription.
- `pricingplanmanager:UpdateSubscription` - Allows principals to modify an existing subscription, such as changing the pricing plan.
- `pricingplanmanager:GetSubscription` – Allows principals read-only access to get details about pricing plan subscriptions.
- `pricingplanmanager:ListSubscriptions` – Allows principals read-only access to list pricing plan subscriptions.
- `pricingplanmanager:ApprovePaidSubscription` – Allows principals to approve a paid subscription and start billing.
- `freetier:GetAccountPlanState` – Allows principals to get the AWS Free Tier plan state for the account, used to determine eligibility for flat-rate pricing plans.
- `cloudwatch:GetMetricData` – Allows principals to retrieve CloudWatch metric data, used to determine eligibility for flat-rate pricing plans.
- `shield:ListProtections` – Allows principals to list AWS Shield protections, used to determine eligibility for flat-rate pricing plans.
- `ec2:DescribeInstances` - Allows principals to get detailed information about instances in Amazon EC2.
- `ec2:DescribeInternetGateways` - Allows principals to get detailed information about internet gateways in Amazon EC2.
- `ec2:DescribeIpamPools` – Allows principals to get detailed information about your IPAM pools.
- `ec2:GetIpamPoolCidrs` – Allows principals to get the CIDRs provisioned to an IPAM pool.

To view the permissions for this policy, see CloudFrontFullAccess in the *AWS Managed Policy Reference*.

> **Note**
>
> The **CloudFrontFullAccess** policy includes the `pricingplanmanager:ApprovePaidSubscription` permission, which commits your account to charges for paid plans that can't be canceled until the end of the current billing period. For automated workflows—such as deployment pipelines, agents, or service roles—we recommend a scoped-down policy that grants `pricingplanmanager:CreateSubscription` without `pricingplanmanager:ApprovePaidSubscription`, so that a human authorizes billing. For more information, see Getting started with the PricingPlanManager API.

> **Important**
>
> If you want CloudFront to create and save access logs, you need to grant additional permissions. For more information, see Permissions.

AWS managed policy: AWSCloudFrontLogger

You can't attach the **AWSCloudFrontLogger** policy to your IAM identities. This policy is attached to a service-linked role that allows CloudFront to perform actions on your behalf. For more information, see the section called "Service-linked roles for Lambda@Edge".

This policy allows CloudFront to push log files to Amazon CloudWatch. For details about the permissions included in this policy, see the section called "Service-linked role permissions for CloudFront logger".

To view the permissions for this policy, see AWSCloudFrontLogger in the *AWS Managed Policy Reference*.

AWS managed policy: AWSLambdaReplicator

You can't attach the **AWSLambdaReplicator** policy to your IAM identities. This policy is attached to a service-linked role that allows CloudFront to perform actions on your behalf. For more information, see the section called "Service-linked roles for Lambda@Edge".

This policy allows CloudFront to create, delete, and disable functions in AWS Lambda to replicate Lambda@Edge functions to AWS Regions. For details about the permissions included in this policy, see the section called "Service-linked role permissions for Lambda replicator".

To view the permissions for this policy, see AWSLambdaReplicator in the *AWS Managed Policy Reference*.

AWS managed policy: AWSCloudFrontVPCOriginServiceRolePolicy

You can't attach the **AWSCloudFrontVPCOriginServiceRolePolicy** policy to your IAM entities. This policy is attached to a service-linked role that allows CloudFront to perform actions on your behalf. For more information, see Use service-linked roles for CloudFront.

This policy allows CloudFront to manage EC2 elastic network interfaces and security groups on your behalf. For details about the permissions included in this policy, see the section called "Service-linked role permissions for CloudFront VPC Origins".

To view the permissions for this policy, see AWSCloudFrontVPCOriginServiceRolePolicy in the *AWS Managed Policy Reference*.

CloudFront updates to AWS managed policies

View details about updates to AWS managed policies for CloudFront since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CloudFront Document history page.

Change	Description	Date
CloudFrontFullAccess – Update to existing policy	CloudFront added new permissions for flat-rate pricing plan eligibility and subscription approval. The new permissions allow principals to use the `pricingplanmanager:ApprovePaidSubscription` , `freetier:GetAccountPlanStat`	July 29, 2026

Change	Description	Date
	e , cloudwatch:GetMetricData , and shield:ListProtections actions.	
CloudFrontReadOnlyAccess – Update to existing policy	CloudFront added new permissions for Amazon EC2. The new permissions allow principals to use the ec2:DescribeIpamPools and ec2:GetIpamPoolCidrs actions.	November 24, 2025
CloudFrontFullAccess – Update to existing policy	CloudFront added new permissions for Amazon EC2. The new permissions allow principals to use the ec2:DescribeIpamPools and ec2:GetIpamPoolCidrs actions.	November 24, 2025
CloudFrontFullAccess – Update to existing policy	CloudFront added a new permission to create an AWS WAF ACL resource, and added create, update, delete, and read permissions to AWS Pricing Plan Manager.	November 18, 2025
CloudFrontFullAccess – Update to existing policy	CloudFront added a new permission to create an AWS WAF ACL resource, and added create, update, delete, and read permissions to AWS Pricing Plan Manager.	November 18, 2025

Change	Description	Date
CloudFrontReadOnlyAccess – Update to existing policy	CloudFront added new permissions for read-only access to AWS Pricing Plan Manager.	November 18, 2025
CloudFrontReadOnlyAccess – Update to existing policy	CloudFront added new permissions for read-only access to AWS Pricing Plan Manager.	November 18, 2025
CloudFrontReadOnlyAccess – Update to existing policy	CloudFront added new permission for ACM. The new permission allows principals to get details about an ACM certificate.	April 28, 2025
CloudFrontFullAccess – Update to existing policy	CloudFront added new permissions for ACM. The new permissions allow principals to get details about an ACM certificate and to request a managed certificate from ACM.	April 28, 2025

Change	Description	Date
CloudFrontFullAccess – Update to existing policy	CloudFront added new permissions for Amazon EC2 and Elastic Load Balancing. The new permissions allow CloudFront to get detailed information about load balancers in Elastic Load Balancing and instances and internet gateways in Amazon EC2.	November 20, 2024
AWSCloudFrontVPCOriginServiceRolePolicy – New policy	CloudFront added a new policy. This policy allows CloudFront to manage EC2 elastic network interfaces and security groups on your behalf.	November 20, 2024
CloudFrontReadOnlyAccess and CloudFrontFullAccess - Update to two existing policies.	CloudFront added new permissions for key value stores. The new permissions allow users to get information about, and take action on, key value stores.	December 19, 2023

Change	Description	Date
CloudFrontReadOnlyAccess – Update to an existing policy	CloudFront added a new permission to describe CloudFront Functions. This permission allows the user, group, or role to read information and metadata about a function, but not the function's code.	September 8, 2021
CloudFront started tracking changes	CloudFront started tracking changes for its AWS managed policies.	September 8, 2021

Use service-linked roles for CloudFront

Amazon CloudFront uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to CloudFront. Service-linked roles are predefined by CloudFront and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up CloudFront easier because you don't have to manually add the necessary permissions. CloudFront defines the permissions of its service-linked roles, and unless defined otherwise, only CloudFront can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your CloudFront resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-linked roles** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for CloudFront VPC Origins

CloudFront VPC Origins uses the service-linked role named **AWSServiceRoleForCloudFrontVPCOrigin** – Allows CloudFront to manage EC2 elastic network interfaces and security groups on your behalf.

The AWSServiceRoleForCloudFrontVPCOrigin service-linked role trusts the following services to assume the role:

- `vpcorigin.cloudfront.amazonaws.com`

The role permissions policy named AWSCloudFrontVPCOriginServiceRolePolicy allows CloudFront VPC Origins to complete the following actions on the specified resources:

- Action: `ec2:CreateNetworkInterface` on `arn:aws:ec2:*:*:network-interface/*`
- Action: `ec2:CreateNetworkInterface` on `arn:aws:ec2:*:*:subnet/*` and `arn:aws:ec2:*:*:security-group/*`
- Action: `ec2:CreateSecurityGroup` on `arn:aws:ec2:*:*:security-group/*`
- Action: `ec2:CreateSecurityGroup` on `arn:aws:ec2:*:*:vpc/*`
- Action: `ec2:ModifyNetworkInterfaceAttribute`, `ec2:DeleteNetworkInterface`, `ec2:DeleteSecurityGroup`, `ec2:AssignIpv6Addresses`, and `ec2:UnassignIpv6Addresses` on `supported AWS resources that have the aws:ResourceTag/aws.cloudfront.vpcorigin tag enabled`
- Action: `ec2:DescribeNetworkInterfaces`, `ec2:DescribeSecurityGroups`, `ec2:DescribeInstances`, `ec2:DescribeInternetGateways`, `ec2:DescribeSubnets`, `ec2:DescribeRegions`, and `ec2:DescribeAddresses` on `all AWS resources that the actions support`
- Action: `ec2:CreateTags` on `arn:aws:ec2:*:*:security-group/*` and `arn:aws:ec2:*:*:network-interface/*`
- Action: `elasticloadbalancing:DescribeLoadBalancers`, `elasticloadbalancing:DescribeListeners`, and `elasticloadbalancing:DescribeTargetGroups` on `all AWS resources that the actions support`

You must configure permissions to allow your users, groups, or roles to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

Create a service-linked role for CloudFront VPC Origins

You don't need to manually create a service-linked role. When you create a VPC origin in the AWS Management Console, the AWS CLI, or the AWS API, CloudFront VPC Origins creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a VPC origin, CloudFront VPC Origins creates the service-linked role for you again.

Edit a service-linked role for CloudFront VPC Origins

CloudFront VPC Origins does not allow you to edit the AWSServiceRoleForCloudFrontVPCOrigin service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Delete a service-linked role for CloudFront VPC Origins

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
>
> If the CloudFront service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete CloudFront VPC Origins resources used by the AWSServiceRoleForCloudFrontVPCOrigin

- Delete the VPC origin resources in your account.
  - It might take some time for CloudFront to finish deleting the resources from your account. If you can't delete the service-linked role right away, wait and try again.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForCloudFrontVPCOrigin service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for CloudFront VPC Origins service-linked roles

CloudFront VPC Origins does not support using service-linked roles in every Region where the service is available. You can use the AWSServiceRoleForCloudFrontVPCOrigin role in the following Regions.

Region name	Region identity	Support in CloudFront
US East (N. Virginia)	eusc-de-east-1	Yes
	eusc-de-east-1	Yes
US West (N. California)	us-west-1 (except AZ usw1-az2)	Yes
US West (Oregon)	us-west-2	Yes
Africa (Cape Town)	af-south-1	Yes
Asia Pacific (Hong Kong)	ap-east-1	Yes
Asia Pacific (Jakarta)	ap-southeast-3	Yes
Asia Pacific (Melbourne)	ap-southeast-4	Yes
Asia Pacific (Mumbai)	ap-south-1	Yes
Asia Pacific (Hyderabad)	ap-south-2	Yes
Asia Pacific (Osaka)	ap-northeast-3	Yes
Asia Pacific (Seoul)	ap-northeast-2	Yes
Asia Pacific (Singapore)	ap-southeast-1	Yes
Asia Pacific (Sydney)	ap-southeast-2	Yes

Region name	Region identity	Support in CloudFront
Asia Pacific (Tokyo)	ap-northeast-1 (except AZ apne1-az3)	Yes
Canada (Central)	ca-central-1 (except AZ cac1-az3)	Yes
Canada West (Calgary)	ca-west-1	Yes
Europe (Frankfurt)	eu-central-1	Yes
Europe (Ireland)	eu-west-1	Yes
Europe (London)	eu-west-2	Yes
Europe (Milan)	eu-south-1	Yes
Europe (Paris)	eu-west-3	Yes
Europe (Spain)	eu-south-2	Yes
Europe (Stockholm)	eu-north-1	Yes
Europe (Zurich)	eu-central-2	Yes
Israel (Tel Aviv)	il-central-1	Yes
Middle East (Bahrain)	me-south-1	Yes
Middle East (UAE)	me-central-1	Yes
South America (São Paulo)	sa-east-1	Yes

Troubleshoot Amazon CloudFront identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CloudFront and IAM.

Topics

- I'm not authorized to perform an action in CloudFront
- I'm not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my CloudFront resources

I'm not authorized to perform an action in CloudFront

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `cloudfront:`*`GetWidget`* permissions.

```
User: arn:aws-eusc:iam::123456789012:user/mateojackson is not authorized to perform:
 cloudfront:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `cloudfront:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I'm not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to CloudFront.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in CloudFront. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws-eusc:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my CloudFront resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether CloudFront supports these features, see How Amazon CloudFront works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Logging and monitoring in Amazon CloudFront

Monitoring is an important part of maintaining the availability and performance of CloudFront and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your CloudFront resources and activity, and responding to potential incidents:

Amazon CloudWatch alarms

Using CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms do not invoke actions when a metric is in a particular state. Rather the state must have changed and been maintained for a specified number of periods.

For more information, see Monitor CloudFront metrics with Amazon CloudWatch.

AWS CloudTrail logs

CloudTrail provides a record of API actions taken by a user, role, or an AWS service in CloudFront. Using the information collected by CloudTrail, you can determine the API request that was made to CloudFront, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information, see Logging Amazon CloudFront API calls using AWS CloudTrail.

CloudFront standard logs and real-time access logs

CloudFront logs provide detailed records about requests that are made to a distribution. These logs are useful for many applications. For example, log information can be useful in security and access audits.

For more information, see Access logs (standard logs) and Create and use real-time access log configurations.

Edge function logs

Logs generated by edge functions, both CloudFront Functions and Lambda@Edge, are sent directly to Amazon CloudWatch Logs and are not stored anywhere by CloudFront. CloudFront Functions uses an AWS Identity and Access Management (IAM) service-linked role to send customer-generated logs directly to CloudWatch Logs in your account.

For more information, see Edge function logs.

CloudFront console reports

The CloudFront console includes a variety of reports, including the cache statistics report, the popular objects report, and the top referrers report. Most CloudFront console reports are based on the data in CloudFront access logs, which contain detailed information about every user request that CloudFront receives. However, you don't need to enable access logs to view the reports.

For more information, see View CloudFront reports in the console.

Compliance validation for Amazon CloudFront

Third-party auditors assess the security and compliance of Amazon CloudFront as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using CloudFront is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance on AWS – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

  The AWS HIPAA compliance program includes CloudFront (excluding content delivery through CloudFront Embedded POPs) as a HIPAA eligible service. If you have an executed Business Associate Addendum (BAA) with AWS, you can use CloudFront (excluding content delivery through CloudFront Embedded POPs) to deliver content that contains protected health information (PHI). For more information, see HIPAA Compliance.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub CSPM – This AWS service uses security controls to evaluate resource configurations and security standards to help you comply with various compliance frameworks. For more information about using Security Hub CSPM to evaluate CloudFront resources, see Amazon CloudFront controls in the *AWS Security Hub CSPM User Guide*.

CloudFront compliance best practices

This section provides best practices and recommendations for compliance when you use Amazon CloudFront to serve your content.

You might run PCI-compliant or HIPAA-compliant workloads that are based on the AWS shared responsibility model. In this case, we recommend that you log your CloudFront usage data for the last 365 days for future auditing purposes. To log usage data, you can do the following:

- Enable CloudFront access logs. For more information, see Access logs (standard logs).
- Capture requests that are sent to the CloudFront API. For more information, see Logging Amazon CloudFront API calls using AWS CloudTrail.

In addition, see the following for details about how CloudFront is compliant with the PCI DSS and SOC standards.

Payment Card Industry Data Security Standard (PCI DSS)

CloudFront (excluding content delivery through CloudFront Embedded POPs) supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with Payment Card Industry (PCI) Data Security Standard (DSS). For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see PCI DSS Level 1.

As a security best practice, we recommend that you don't cache credit card information in CloudFront edge caches. For example, you can configure your origin to include a `Cache-Control:no-cache="`*`field-name`*`"` header in responses that contain credit card information, such as the last four digits of a credit card number and the card owner's contact information.

System and Organization Controls (SOC)

CloudFront (excluding content delivery through CloudFront Embedded POPs) is compliant with System and Organization Controls (SOC) measures, including SOC 1, SOC 2, and SOC 3. SOC reports are independent, third-party examination reports that demonstrate how AWS achieves key compliance controls and objectives. These audits make sure that the appropriate safeguards and procedures are in place to protect against risks that might affect the security, confidentiality, and availability of customer and company data. The results of these third-party audits are available on the AWS SOC Compliance website, where you can view the published reports to get more information about the controls that support AWS operations and compliance.

Resilience in Amazon CloudFront

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability

Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

CloudFront origin failover

In addition to the support of AWS global infrastructure, Amazon CloudFront offers an *origin failover* feature to help support your data resiliency needs. CloudFront is a global service that delivers your content through a worldwide network of data centers called *edge locations* or *points of presence* (POPs). If your content is not already cached in an edge location, CloudFront retrieves it from an origin that you've identified as the source for the definitive version of the content.

You can improve resiliency and increase availability for specific scenarios by setting up CloudFront with origin failover. To get started, you create an origin group in which you designate a primary origin for CloudFront plus a second origin. CloudFront automatically switches to the second origin when the primary origin returns specific HTTP status code failure responses. For more information, see Optimize high availability with CloudFront origin failover.

Infrastructure security in Amazon CloudFront

As a managed service, Amazon CloudFront is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access CloudFront through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

CloudFront Functions uses a highly secure isolation barrier between AWS accounts, making sure that customer environments are secure against side-channel attacks like Spectre and Meltdown. Functions cannot access or modify data belonging to other customers. Functions run in a dedicated single-threaded process on a dedicated CPU without hyperthreading. In any given CloudFront edge

location point of presence (POP), CloudFront Functions only serves one customer at a time, and all customer-specific data is cleared between function executions.

Troubleshooting

Use this section to troubleshoot common problems you might encounter when you set up Amazon CloudFront to distribute your content.

Each topic provides detailed guidance on identifying the root cause of common issues and step-by-step instructions to resolve them.

Topics

- Troubleshooting distribution issues
- Troubleshooting error response status codes in CloudFront
- Load testing CloudFront

Troubleshooting distribution issues

Use the information here to help you diagnose and fix certificate errors, access-denied issues, or other common issues that you might encounter when setting up your website or application with Amazon CloudFront distributions.

Topics

- CloudFront returns an Access Denied error
- CloudFront returns an InvalidViewerCertificate error when I try to add an alternate domain name
- CloudFront returns an incorrectly configured DNS record error when I try to add a new CNAME
- I can't view the files in my distribution
- Error message: Certificate: <certificate-id> is being used by CloudFront

CloudFront returns an Access Denied error

If you're using an Amazon S3 bucket as the origin for your CloudFront distribution, you might see an Access Denied (403) error message in the following examples.

Contents

- You specified a missing object from the Amazon S3 origin

- Your Amazon S3 origin is missing IAM permissions
- You're using invalid credentials or don't have sufficient permissions

You specified a missing object from the Amazon S3 origin

Verify that the requested object in your bucket exists. Object names are case sensitive. Entering an invalid object name can return an access denied error code.

For example, if you follow the CloudFront tutorial to create a basic distribution, you create an Amazon S3 bucket as the origin and upload an example `index.html` file.

In your web browser, if you enter `https://d111111abcdef8.cloudfront.net/INDEX.HTML` instead of `https://d111111abcdef8.cloudfront.net/index.html`, you might see a similar message because the `index.html` file in the URL path is case sensitive.

```
<Error>
<Code>AccessDenied</Code>
<Message>Access Denied</Message>
<RequestId>22Q367AHT7Y1ABCD</RequestId>
<HostId>
ABCDE/Vg+7PSNa/d/IfFQ8Fb92TGQ0KH0ZwG5iEKbc6+e06DdMS1ZW+ryB9GFRIVtS66rSSy6So=
</HostId>
</Error>
```

Your Amazon S3 origin is missing IAM permissions

Verify that you've selected the correct Amazon S3 bucket as the origin domain and name. The origin (Amazon S3) must have the correct permissions.

If you don't specify the correct permissions, an `AccessDenied` message can appear for your viewers.

When you distribute content from Amazon S3 and you're also using AWS Key Management Service (AWS KMS) server-side encryption (SSE-KMS), there are additional IAM permissions that you need to specify for the KMS key and Amazon S3 bucket. Your CloudFront distribution needs these permissions to use the KMS key, which is used for encryption of the origin Amazon S3 bucket.

The configurations to the Amazon S3 bucket policy allow the CloudFront distribution to retrieve the encrypted objects for content delivery.

To verify your Amazon S3 bucket and KMS key permissions

1. Verify that the KMS key that you're using is the same key that your Amazon S3 bucket uses for default encryption. For more information, see Specifying server-side encryption with AWS KMS (SSE-KMS) in the *Amazon Simple Storage Service User Guide*.
2. Verify that the objects in the bucket are encrypted with the same KMS key. You can select any object from the Amazon S3 bucket and check the server-side encryption settings to verify the KMS key ARN.
3. Edit the Amazon S3 bucket policy to grant CloudFront permission to call the `GetObject` API operation from the Amazon S3 bucket. For an example Amazon S3 bucket policy that uses origin access control, see Grant CloudFront permission to access the S3 bucket.
4. Edit the KMS key policy to grant CloudFront permission to perform the actions to `Encrypt`, `Decrypt`, and `GenerateDataKey*`. To align with least privilege permission, specify a `Condition` element so that only the specified CloudFront distribution can perform the listed actions. You can customize the policy for your existing AWS KMS policy. For an example KMS key policy, see the SSE-KMS.

If you're using origin access identity (OAI) instead of OAC, the permissions to the Amazon S3 bucket are slightly different because you grant permission to an identity instead of the AWS service. For more information, see Give an origin access identity permission to read files in the Amazon S3 bucket.

If you still can't view your files in your distribution, see I can't view the files in my distribution.

You're using invalid credentials or don't have sufficient permissions

An Access Denied error message can appear if you're using incorrect or expired AWS SCT credentials (access key and secret key) or your IAM role or user is missing a required permission to perform an action on a CloudFront resource. For more information about access denied error messages, see Troubleshooting access denied error messages in the *IAM User Guide*.

For information about how IAM works with CloudFront, see Identity and Access Management for Amazon CloudFront.

CloudFront returns an InvalidViewerCertificate error when I try to add an alternate domain name

If CloudFront returns an `InvalidViewerCertificate` error when you try to add an alternate domain name (CNAME) to your distribution, review the following information to help troubleshoot the problem. This error can indicate that one of the following issues must be resolved before you can successfully add the alternate domain name.

The following errors are listed in the order in which CloudFront checks for authorization to add an alternate domain name. This can help you troubleshoot issues because based on the error that CloudFront returns, you can tell which verification checks have completed successfully.

There's no certificate attached to your distribution.

To add an alternate domain name (CNAME), you must attach a trusted, valid certificate to your distribution. Please review the requirements, obtain a valid certificate that meets them, attach it to your distribution, and then try again. For more information, see Requirements for using alternate domain names.

There are too many certificates in the certificate chain for the certificate that you've attached.

You can only have up to five certificates in a certificate chain. Reduce the number of certificates in the chain, and then try again.

The certificate chain includes one or more certificates that aren't valid for the current date.

The certificate chain for a certificate that you have added has one or more certificates that aren't valid, either because a certificate isn't valid yet or a certificate has expired. Check the **Not Valid Before** and **Not Valid After** fields in the certificates in your certificate chain to make sure that all of the certificates are valid based on the dates that you've listed.

The certificate that you've attached isn't signed by a trusted Certificate Authority (CA).

The certificate that you attach to CloudFront to verify an alternate domain name cannot be a self-signed certificate. It must be signed by a trusted CA. For more information, see Requirements for using alternate domain names.

The certificate that you've attached isn't formatted correctly

The domain name and IP address format that are included in the certificate, and the format of the certificate itself, must follow the standard for certificates.

There was a CloudFront internal error.

CloudFront was blocked by an internal issue and couldn't make validation checks for certificates. In this scenario, CloudFront returns an HTTP 500 status code and indicates that there is an internal CloudFront problem with attaching the certificate. Wait a few minutes, and then try again to add the alternate domain name with the certificate.

The certificate that you've attached doesn't cover the alternate domain name that you're trying to add.

For each alternate domain name that you add, CloudFront requires that you attach a valid SSL/TLS certificate from a trusted Certificate Authority (CA) that covers the domain name, to validate your authorization to use it. Please update your certificate to include a domain name that covers the CNAME that you're trying to add. For more information and examples of using domain names with wildcards, see Requirements for using alternate domain names.

CloudFront returns an incorrectly configured DNS record error when I try to add a new CNAME

When you have an existing wildcard DNS entry pointing to a CloudFront distribution, if you try to add a new CNAME with a more specific name, you might encounter the following error:

```
One or more aliases specified for the distribution includes an incorrectly configured
 DNS record that points to another CloudFront distribution. You must update the DNS
 record to correct the problem.
```

This error occurs because CloudFront queries the DNS against the CNAME and the wildcard DNS entry resolves to another distribution.

To resolve this, first create another distribution, then create a DNS entry pointing to the new distribution. Finally, add the more specific CNAME. For more information on how to add CNAMEs, see Add an alternate domain name.

I can't view the files in my distribution

If you can't view the files in your CloudFront distribution, see the following topics for some common solutions.

Did you sign up for both CloudFront and Amazon S3?

To use Amazon CloudFront with an Amazon S3 origin, you must sign up for both CloudFront and Amazon S3, separately. For more information about signing up for CloudFront and Amazon S3, see Set up your AWS account.

Are your Amazon S3 bucket and object permissions set correctly?

If you're using CloudFront with an Amazon S3 origin, the original versions of your content are stored in an S3 bucket. To serve the content to your viewers, we recommend that you use CloudFront Origin Access Control (OAC) to secure Amazon S3 bucket access. This means your S3 bucket is reachable only through CloudFront. OAC controls viewer access and secure delivery by using CloudFront. For more information about OAC, see the section called “Restrict access to an Amazon S3 origin”.

For more information about managing your bucket access, see Blocking public access to your Amazon S3 storage in the *Amazon S3 User Guide*.

Object properties and bucket properties are independent. You must explicitly grant privileges to each object in Amazon S3. Objects do not inherit properties from buckets, and object properties must be set independently of the bucket.

Is your alternate domain name (CNAME) correctly configured?

If you already have an existing CNAME record for your domain name, update that record or replace it with a new one that points to your distribution's domain name.

Also, make sure that your CNAME record points to your distribution's domain name, not your Amazon S3 bucket. You can confirm that the CNAME record in your DNS system points to your distribution's domain name. To do so, use a DNS tool like **dig**.

The following example shows a dig request for a domain name called `images.example.com` and the relevant part of the response. Under `ANSWER SECTION`, see the line that contains `CNAME`. The CNAME record for your domain name is set up correctly if the value on the right side of CNAME is your CloudFront distribution's domain name. If it's your Amazon S3 origin server bucket or some other domain name, then the CNAME record is set up incorrectly.

```
[prompt]> dig images.example.com
```

```
; <<> DiG 9.3.3rc2 <<> images.example.com
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15917
;; flags: qr rd ra; QUERY: 1, ANSWER: 9, AUTHORITY: 2, ADDITIONAL: 0
;; QUESTION SECTION:
;images.example.com.   IN  A
;; ANSWER SECTION:
images.example.com. 10800 IN CNAME d111111abcdef8.cloudfront.net.
...
...
```

For more information about CNAMEs, see Use custom URLs by adding alternate domain names (CNAMEs).

Are you referencing the correct URL for your CloudFront distribution?

Make sure that the URL that you're referencing uses the domain name (or CNAME) of your CloudFront distribution, not your Amazon S3 bucket or custom origin.

Do you need help troubleshooting a custom origin?

If you need AWS to help you troubleshoot a custom origin, AWS will probably need to inspect the `X-Amz-Cf-Id` header entries from your requests. If you are not already logging these entries, you might want to consider it for the future. For more information, see the section called “Use Amazon EC2 (or another custom origin)”. For further help, see the AWS Support Center.

Error message: Certificate: <certificate-id> is being used by CloudFront

Problem: You're trying to delete an SSL/TLS certificate from the IAM certificate store, and you're getting the message "Certificate: <certificate-id> is being used by CloudFront."

Solution: Every CloudFront distribution must be associated either with the default CloudFront certificate or with a custom SSL/TLS certificate. Before you can delete an SSL/TLS certificate, you must either rotate the certificate (replace the current custom SSL/TLS certificate with another custom SSL/TLS certificate) or revert from using a custom SSL/TLS certificate to using the default CloudFront certificate. To fix that, complete the steps in one of the following procedures:

- Rotate SSL/TLS certificates
- Revert from a custom SSL/TLS certificate to the default CloudFront certificate

Troubleshooting error response status codes in CloudFront

If CloudFront requests an object from your origin, and the origin returns an HTTP 4xx or 5xx status code, there's a problem with communication between CloudFront and your origin.

This topic also includes troubleshooting steps for these status codes when using Lambda@Edge or CloudFront Functions.

The following topics provide detailed explanations of the potential causes behind these error responses and offer step-by-step guidance on how to diagnose and resolve the underlying issues.

Topics

- HTTP 400 status code (Bad Request)
- HTTP 401 status code (Unauthorized)
- HTTP 403 status code (Invalid method)
- HTTP 403 status code (Permission Denied)
- HTTP 404 status code (Not Found)
- HTTP 412 status code (Precondition Failed)
- HTTP 500 status code (Internal Server Error)
- HTTP 502 status code (Bad Gateway)
- HTTP 503 status code (Service Unavailable)
- HTTP 504 status code (Gateway Timeout)

HTTP 400 status code (Bad Request)

CloudFront returns a 400 bad request when the client sends some invalid data in the request such as missing or incorrect content in the payload or parameters. This could also represent a generic client error.

Amazon S3 origin returns a 400 error

If you're using an Amazon S3 origin with your CloudFront distribution, your distribution might send error responses with HTTP status code 400 Bad Request, and a message similar to the following:

The authorization header is malformed; the region '<AWS Region>' is wrong; expecting '<AWS Region>'

For example:

The authorization header is malformed; the region 'us-east-1' is wrong; expecting 'us-west-2'

This problem can occur in the following scenario:

1. Your CloudFront distribution's origin is an Amazon S3 bucket.
2. You moved the S3 bucket from one AWS Region to another. That is, you deleted the S3 bucket, then later you created a new bucket with the same bucket name, but in a different AWS Region than where the original S3 bucket was located.

To fix this error, update your CloudFront distribution so that it finds the S3 bucket in the bucket's current AWS Region.

To update your CloudFront distribution

1. Sign in to the AWS Management Console and open the CloudFront console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudfront/v4/home.
2. Choose the distribution that produces this error.
3. Choose **Origins and Origin Groups**.
4. Find the origin for the S3 bucket that you moved. Select the check box next to this origin, then choose **Edit**.
5. Choose **Yes, Edit**. You do not need to change any settings before choosing **Yes, Edit**.

When you complete these steps, CloudFront redeploys your distribution. While the distribution is deploying, you see the **Deploying** status under the **Last modified** column. Some time after the deployment is complete, you should stop receiving the `AuthorizationHeaderMalformed` error responses.

Application Load Balancer origin returns a 400 error

If you're using an Application Load Balancer origin with your CloudFront distribution, possible causes of a 400 error include the following:

- The client sent a malformed request that does not meet the HTTP specification.
- The request header exceeds 16 KB per request line, 16 KB per single header, or 64 KB for the entire request header.

- The client closed the connection before sending the full request body.

HTTP 401 status code (Unauthorized)

A 401 Unauthorized response status code indicates that the client request hasn't been completed because it lacks valid authentication credentials for the requested resource. This status code is sent with an HTTP `WWW-Authenticate` response header that contains information about how the client can request the resource again after prompting the user for authentication credentials. For more information, see 401 Unauthorized.

In CloudFront, if your origin expects an `Authorization` header to authenticate the requests, CloudFront needs to forward the `Authorization` header to the origin to avoid a 401 Unauthorized error. When CloudFront forwards a viewer request to your origin, CloudFront removes some viewer headers by default, including the `Authorization` header. To make sure that your origin always receives the `Authorization` header in origin requests, you have the following options:

- Add the `Authorization` header to the cache key using a cache policy. All headers in the cache key are automatically included in origin requests. For more information, see Control the cache key with a policy.
- Add the `Authorization` header individually in an origin request policy. For more information, see Control origin requests with a policy.
- Use an origin request policy that forwards all viewer headers to the origin. CloudFront provides a managed origin request policy for this use case, called **Managed-AllViewer**. For more information, see Use managed origin request policies.

> **Important**
>
> If you forward the `Authorization` header to your origin without including it in the cache key, make sure that your origin does not rely on the `Authorization` header for access control of cached content. When the `Authorization` header is not part of the cache key, CloudFront can serve the same cached response to both authorized and unauthorized viewers. Either include the `Authorization` header in the cache key using a cache policy, or disable caching entirely for origins that require origin-side authorization processing.

For more information, see How can I configure CloudFront to forward the Authorization header to the origin?

HTTP 403 status code (Invalid method)

CloudFront returns a 403 (Invalid method) error if you're trying to use an HTTP method that you haven't specified in the CloudFront distribution. You can specify one of the following options for your distribution:

- CloudFront forwards only `GET` and `HEAD` requests.
- CloudFront forwards only `GET`, `HEAD`, and `OPTIONS` requests.
- CloudFront forwards `GET`, `HEAD`, `OPTIONS`, `PUT`, `PATCH`, `POST`, and `DELETE` requests. (If you select this option, you might need to restrict access to your Amazon S3 bucket or custom origin so that users can't perform operations that you don't want them to. For example, you might not want users to have permissions to delete objects from your origin.

HTTP 403 status code (Permission Denied)

An HTTP 403 error means the client isn't authorized to access the requested resource. The client understands the request, but can't authorize viewer access. The following are common causes when CloudFront returns this status code:

Topics

- Alternate CNAME is incorrectly configured
- AWS WAF is configured on CloudFront distribution or at the origin
- Custom origin returns a 403 error
- Amazon S3 origin returns a 403 error
- Geographic restrictions return a 403 error
- Signed URL or signed cookie configuration returns a 403 error
- Stacked distributions cause a 403 error

Alternate CNAME is incorrectly configured

Verify that you specified the correct CNAME for your distribution. To use an alternate CNAME instead of the default CloudFront URL:

1. Create a CNAME record in your DNS to point the CNAME to CloudFront distribution URL.
2. Add the CNAME in your CloudFront distribution configuration.

If you create the DNS record but don't add the CNAME in your CloudFront distribution configuration, then the request returns a 403 error. For more information about configuring a custom CNAME, see Use custom URLs by adding alternate domain names (CNAMEs).

AWS WAF is configured on CloudFront distribution or at the origin

When AWS WAF sits between the client and CloudFront, CloudFront can't distinguish between a 403 error code that's returned by your origin and a 403 error code that's returned by AWS WAF when a request is blocked.

To find the source of the 403 status code, check your AWS WAF web access control list (ACL) rule for a blocked request. For more information, see the following topics:

- AWS WAF web access control lists (web ACLs)
- Testing and tuning your AWS WAF protections

Custom origin returns a 403 error

If you're using a custom origin, you might see a 403 error if you have a custom firewall configuration at the origin. To troubleshoot, make the request directly to the origin. If you can replicate the error without CloudFront, then the origin is causing the 403 error.

If the custom origin is causing the error, check the origin logs to identify what might be causing the error. For more information, see the following troubleshooting topics:

- How do I troubleshoot HTTP 403 errors from API Gateway?
- How do I troubleshoot Application Load Balancer HTTP 403 forbidden errors?

Amazon S3 origin returns a 403 error

You might see a 403 error for the following reasons:

- CloudFront doesn't have access to the Amazon S3 bucket. This can happen if origin access identity (OAI) or origin access control (OAC) isn't enabled for your distribution and the bucket is private.

- The specified path in the requested URL isn't correct.
- The requested object doesn't exist.
- The host header was forwarded with the REST API endpoint. For more information, see HTTP Host header bucket specification in the *Amazon Simple Storage Service User Guide*.
- You configured custom error pages. For more information, see How CloudFront processes errors when you have configured custom error pages.

Geographic restrictions return a 403 error

If you enabled geographic restrictions (also known as geoblocking) to prevent users in specific geographic locations from accessing content that you're distributing through a CloudFront distribution, blocked users receive a 403 error.

For more information, see Restrict the geographic distribution of your content.

Signed URL or signed cookie configuration returns a 403 error

If you enabled **Restrict viewer access** for your distribution's behavior configuration, then requests that don't use signed cookies or signed URLs result in a 403 error. For more information, see the following topics:

- Serve private content with signed URLs and signed cookies
- How do I troubleshoot issues related to a signed URL or signed cookies in CloudFront?

Stacked distributions cause a 403 error

If you have two or more distributions within a chain of requests to the origin endpoint, CloudFront returns a 403 error. We don't recommend placing one distribution in front of another.

HTTP 404 status code (Not Found)

CloudFront returns a 404 (Not Found) error when the client attempts to access a resource that doesn’t exist. If you receive this error with your CloudFront distribution, common causes include the following:

- Resource doesn't exist.
- URL is incorrect.

- Custom origin returning a 404.
- Custom error pages returning a 404. (Any error code might be translated to 404.) For more information, see How CloudFront processes errors when you have configured custom error pages.
- Custom error page accidentally deleted, resulting in a 404 because the request looks for the deleted custom error page. For more information, see How CloudFront processes errors if you haven't configured custom error pages.
- Incorrect origin path. If the origin path is populated, its value is appended to the path of each request from the browser before the request is forwarded to the origin. For more information, see Origin path.

HTTP 412 status code (Precondition Failed)

CloudFront returns a 412 (Precondition Failed) error code when access to the target resource has been denied. In some cases, a server is configured to accept requests only after certain conditions are fulfilled. If any of the specified conditions are not met, then the server doesn't allow the client to access the given resource. Instead, the server responds with a 412 error code.

Common causes of a 412 error in CloudFront include:

- Conditional requests on methods other than `GET` or `HEAD` when the condition defined by the `If-Unmodified-Since` or `If-None-Match` headers is not fulfilled. In that case, the request, usually an upload or a modification of a resource, can't be made.
- A condition in one or more of the request fields in the CloudFront UpdateDistribution API operation evaluates as false.

HTTP 500 status code (Internal Server Error)

An HTTP 500 status code (Internal Server Error) indicates that the server encountered an unexpected condition that prevented it from fulfilling the request. The following are some common causes of 500 errors in Amazon CloudFront.

Topics

- Origin server returns 500 error to CloudFront

Origin server returns 500 error to CloudFront

Your origin server might be returning a 500 error to CloudFront. Refer to the following troubleshooting topics for more information:

- **If Amazon S3 returns a 500 error**, see How do I troubleshoot an HTTP 500 or 503 error from Amazon S3?
- **If API Gateway returns a 500 error**, see How do I troubleshoot 5xx errors for API Gateway REST API?.
- **If Elastic Load Balancing returns a 500 error**, see HTTP 500: Internal server error in the *User Guide for Application Load Balancers*.

If the preceding list doesn't resolve the 500 error, the issue might be with a CloudFront Point of Presence returning an internal server error. You can contact Support for assistance.

HTTP 502 status code (Bad Gateway)

CloudFront returns an HTTP 502 status code (Bad Gateway) when CloudFront wasn't able to serve the requested object because it couldn't connect to the origin server.

If you're using Lambda@Edge, the issue might be a Lambda validation error. If you receive an HTTP 502 error with the `NonS3OriginDnsError` error code, there's likely a DNS configuration problem that's preventing CloudFront from connecting to the origin.

Topics

- SSL/TLS negotiation failure between CloudFront and a custom origin server
- Origin is not responding with supported ciphers/protocols
- SSL/TLS certificate on the origin is expired, invalid, self-signed, or the certificate chain is in the wrong order
- Origin is not responding on specified ports in origin settings
- Lambda validation error
- CloudFront function validation error
- DNS error (NonS3OriginDnsError)
- Application Load Balancer origin 502 error
- API Gateway origin 502 error

SSL/TLS negotiation failure between CloudFront and a custom origin server

If you use a custom origin that requires HTTPS between CloudFront and your origin, mismatched domain names might cause errors. The SSL/TLS certificate on your origin *must* include a domain name that matches either the **Origin Domain** that you specified for the CloudFront distribution or the `Host` header of the origin request.

If the domain names don't match, the SSL/TLS handshake fails, and CloudFront returns an HTTP status code 502 (Bad Gateway) and sets the `X-Cache` header to `Error from cloudfront`.

To determine whether domain names in the certificate match the **Origin Domain** in the distribution or the `Host` header, you can use an online SSL checker or OpenSSL. If the domain names don't match, you have two options:

- Get a new SSL/TLS certificate that includes the applicable domain names.

  If you use AWS Certificate Manager (ACM), see Requesting a public certificate in the *AWS Certificate Manager User Guide* to request a new certificate.
- Change the distribution configuration so CloudFront no longer tries to use SSL to connect with your origin.

Online SSL checker

To find an SSL test tool, search the internet for "online ssl checker." Typically, you specify the name of your domain, and the tool returns a variety of information about your SSL/TLS certificate. Confirm that the certificate contains your domain name in the **Common Name** or **Subject Alternative Names** fields.

OpenSSL

To help troubleshoot HTTP 502 errors from CloudFront, you can use OpenSSL to try to make an SSL/TLS connection to your origin server. If OpenSSL is not able to make a connection, that can indicate a problem with your origin server's SSL/TLS configuration. If OpenSSL is able to make a connection, it returns information about the origin server's certificate, including the certificate's common name (`Subject CN` field) and subject alternative name (`Subject Alternative Name` field).

Use the following OpenSSL command to test the connection to your origin server (replace *`origin domain`* with your origin server's domain name, such as example.com):

```
openssl s_client -connect origin domain name:443
```

If the following are true:

- Your origin server supports multiple domain names with multiple SSL/TLS certificates
- Your distribution is configured to forward the `Host` header to the origin

Then add the `-servername` option to the OpenSSL command, as in the following example (replace *CNAME* with the CNAME that's configured in your distribution):

```
openssl s_client -connect origin domain name:443 -servername CNAME
```

Origin is not responding with supported ciphers/protocols

CloudFront connects to origin servers using ciphers and protocols. For a list of the ciphers and protocols that CloudFront supports, see the section called “Supported protocols and ciphers between CloudFront and the origin”. If your origin does not respond with one of these ciphers or protocols in the SSL/TLS exchange, CloudFront fails to connect. You can validate that your origin supports the ciphers and protocols by using an online tool such as SSL Labs. Type the domain name of your origin in the **Hostname** field, and then choose **Submit**. Review the **Common names** and **Alternative names** fields from the test to see if they match your origin's domain name. After the test is finished, find the **Protocols** and **Cipher Suites** sections in the test results to see which ciphers or protocols are supported by your origin. Compare them with the list of the section called “Supported protocols and ciphers between CloudFront and the origin”.

SSL/TLS certificate on the origin is expired, invalid, self-signed, or the certificate chain is in the wrong order

If the origin server returns the following, CloudFront drops the TCP connection, returns HTTP status code 502 (Bad Gateway), and sets the `X-Cache` header to `Error from cloudfront`:

- An expired certificate
- Invalid certificate
- Self-signed certificate
- Certificate chain in the wrong order

> **Note**
>
> If the full chain of certificates, including the intermediate certificate, is not present, CloudFront drops the TCP connection.

For information about installing an SSL/TLS certificate on your custom origin server, see the section called "Require HTTPS to a custom origin".

Origin is not responding on specified ports in origin settings

When you create an origin on your CloudFront distribution, you can set the ports that CloudFront connects to the origin with for HTTP and HTTPS traffic. By default, these are TCP 80/443. You have the option to modify these ports. If your origin is rejecting traffic on these ports for any reason, or if your backend server isn't responding on the ports, CloudFront will fail to connect.

To troubleshoot these issues, check any firewalls running in your infrastructure and validate that they are not blocking the supported IP ranges. For more information, see AWS IP address ranges in the *Amazon VPC User Guide*. Additionally, verify whether your web server is running on the origin.

Lambda validation error

If you're using Lambda@Edge, an HTTP 502 status code can indicate that your Lambda function response was incorrectly formed or included invalid content. For more information about troubleshooting Lambda@Edge errors, see Test and debug Lambda@Edge functions.

CloudFront function validation error

If you're using CloudFront functions, an HTTP 502 status code can indicate that the CloudFront function is trying to add, delete, or change a read-only header. This error does not show up during testing, but will show up after you deploy the function and run the request. To resolve this error, check and update your CloudFront function. For more information, see Update functions.

DNS error (`NonS3OriginDnsError`)

An HTTP 502 error with the `NonS3OriginDnsError` error code indicates that there's a DNS configuration problem that prevents CloudFront from connecting to the origin. If you get this error from CloudFront, make sure that the origin's DNS configuration is correct and working.

When CloudFront receives a request for an object that's expired or is not in its cache, it makes a request to the origin to get the object. To make a successful request to the origin, CloudFront performs a DNS resolution on the origin domain. If the DNS service for your domain is experiencing issues, CloudFront can't resolve the domain name to get the IP address, which results in an HTTP 502 error (`NonS30riginDnsError`). To fix this problem, contact your DNS provider, or, if you are using Amazon Route 53, see Why can't I access my website that uses Route 53 DNS services?

To further troubleshoot this issue, make sure that the authoritative name servers of your origin's root domain or zone apex (such as `example.com`) are functioning correctly. You can use the following commands to find the name servers for your apex origin, with a tool such as dig or nslookup:

```
dig OriginAPEXDomainName NS +short
```

```
nslookup -query=NS OriginAPEXDomainName
```

When you have the names of your name servers, use the following commands to query the domain name of your origin against them to make sure that each responds with an answer:

```
dig OriginDomainName @NameServer
```

```
nslookup OriginDomainName NameServer
```

Important

Make sure that you perform this DNS troubleshooting using a computer that's connected to the public internet. CloudFront resolves the origin domain using public DNS on the internet, so it's important to troubleshoot in a similar context.

If your origin is a subdomain whose DNS authority is delegated to a different name server than the root domain, make sure that the name server (NS) and start of authority (SOA) records are configured correctly for the subdomain. You can check for these records using commands similar to the preceding examples.

For more information about DNS, see Domain Name System (DNS) concepts in the Amazon Route 53 documentation.

Application Load Balancer origin 502 error

If you use Application Load Balancer as your origin and receive a 502 error, see How do I troubleshoot Application Load Balancer HTTP 502 errors?.

API Gateway origin 502 error

If you use API Gateway and receive a 502 error, see How do I resolve HTTP 502 errors from API Gateway REST APIs with Lambda proxy integration?.

HTTP 503 status code (Service Unavailable)

An HTTP 503 status code (Service Unavailable) typically indicates a performance issue on the origin server. In rare cases, it indicates that CloudFront temporarily can't satisfy a request because of resource constraints at an edge location.

If you are using Lambda@Edge or CloudFront Functions, the issue might be an execution error or a Lambda@Edge limit exceeded error.

If you are using origin mutual TLS (origin mTLS), the issue might be due to repeated failed attempts to establish a connection with a specific certificate.

Topics

- Origin server does not have enough capacity to support the request rate
- CloudFront caused the error due to resource constraints at the edge location
- CloudFront caused the error for an origin configured with mutual TLS
- Lambda@Edge or CloudFront Function execution error
- Lambda@Edge limit exceeded

Origin server does not have enough capacity to support the request rate

When an origin server is unavailable or unable to serve incoming requests, it returns an HTTP 503 status code (Service Unavailable). CloudFront then relays the error back to the user. To resolve this issue, try the following solutions:

- **If you use Amazon S3 as your origin server**:
  - You can send 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per partitioned Amazon S3 prefix. When Amazon S3 returns a 503 Slow Down response, this typically indicates an excessive request rate against a specific Amazon S3 prefix.

Because request rates apply per prefix in an S3 bucket, objects should be distributed across multiple prefixes. As the request rate on the prefixes gradually increases, Amazon S3 scales up to handle requests for each of the prefixes separately. As a result, the overall request rate that the bucket handles is a multiple of the number of prefixes.

  - For more information, see Optimizing Amazon S3 performance in the *Amazon Simple Storage Service User Guide*.
- **If you use Elastic Load Balancing as your origin server**:
  - Make sure that your backend instances can respond to health checks.
  - Make sure that your load balancer and backend instances can handle the load.

  For more information, see:
  - How do I troubleshoot 503 errors returned while using Classic Load Balancer?
  - How do I troubleshoot 503 (service unavailable) errors from my Application Load Balancer?
- **If you use a custom origin**:
  - Examine the application logs to make sure that your origin has sufficient resources, such as memory, CPU, and disk size.
  - If you use Amazon EC2 as the backend, make sure that the instance type has the appropriate resources to fulfill the incoming requests. For more information, see Instance types in the *Amazon EC2 User Guide*.
- **If you use API Gateway**:
  - This error is related to the backend integration when the API Gateway API is unable to receive a response. The backend server might be:
    - Overloaded beyond capacity and unable to process new client requests.
    - Under temporary maintenance.
  - To resolve this error, look at your API Gateway application logs to determine if there is an issue with backend capacity, integration, or something else.

CloudFront caused the error due to resource constraints at the edge location

You will receive this error in the rare situation that CloudFront can't route requests to the next best available edge location, and so can't satisfy a request. This error is common when you perform load testing on your CloudFront distribution. To help prevent this, follow the the section called “Load testing CloudFront” guidelines for avoiding 503 (capacity exceeded) errors.

If this happens in your production environment, contact Support.

CloudFront caused the error for an origin configured with mutual TLS

If your distribution uses mutual TLS (mTLS) to connect to a custom origin, CloudFront might return an HTTP 503 status code when it is unable to establish connections to the origin. This can happen when the origin repeatedly fails to complete TLS handshakes or is unreachable due to DNS or connectivity issues. This condition is temporary — CloudFront automatically resumes normal traffic after the origin is reachable again.

To resolve this issue:

- Confirm that the correct client certificate is associated with your distribution and that it has not expired. If you recently updated the certificate, allow time for the change to propagate before it takes effect.
- Confirm that the origin is reachable and that DNS records resolve correctly.

If the issue persists despite configuring the proper certificate in your production environment, contact Support.

Lambda@Edge or CloudFront Function execution error

If you're using Lambda@Edge or CloudFront Functions, an HTTP 503 status code can indicate that your function returned an execution error.

For more details about how to identify and resolve Lambda@Edge errors, see Test and debug Lambda@Edge functions.

For more information about testing CloudFront Functions, see Test functions.

Lambda@Edge limit exceeded

If you're using Lambda@Edge, an HTTP 503 status code can indicate that Lambda returned an error. The error might be caused by one of the following:

- The number of function executions exceeded one of the quotas that Lambda sets to throttle executions in an AWS Region (concurrent executions or invocation frequency).
- The function exceeded the Lambda function timeout quota.

For more information about the Lambda@Edge quotas, see Quotas on Lambda@Edge. For more details about how to identify and resolve Lambda@Edge errors, see the section called "Test and debug". You can also see the Lambda service quotas in the *AWS Lambda Developer Guide*.

HTTP 504 status code (Gateway Timeout)

An HTTP 504 status code (gateway timeout) indicates that when CloudFront forwarded a request to the origin (because the requested object wasn't in the edge cache), one of the following happened:

- The origin returned an HTTP 504 status code to CloudFront.
- The origin didn't respond before the request expired.

CloudFront will return an HTTP 504 status code if traffic is blocked to the origin by a firewall or security group, or if the origin isn't accessible on the internet. Check for those issues first. Then, if access isn't the problem, explore application delays and server timeouts to help you identify and fix the issues.

Topics

- Configure the firewall on your origin server to allow CloudFront traffic
- Configure the security groups on your origin server to allow CloudFront traffic
- Make your custom origin server accessible on the internet
- Find and fix delayed responses from applications on your origin server

Configure the firewall on your origin server to allow CloudFront traffic

If the firewall on your origin server blocks CloudFront traffic, CloudFront returns an HTTP 504 status code, so it's good to make sure that isn't the issue before checking for other problems.

The method that you use to determine if this is an issue with your firewall depends on what system your origin server uses:

- If you use an IPTable firewall on a Linux server, you can search for tools and information to help you work with IPTables.
- If you use Windows Firewall on a Windows server, see Add or Edit Firewall Rule in the Microsoft documentation.

When you evaluate the firewall configuration on your origin server, look for any firewalls or security rules that block traffic from CloudFront edge locations, based on the published IP address range. For more information, see Locations and IP address ranges of CloudFront edge servers.

If the CloudFront IP address range is allowed to connect to your origin server, make sure to update your server's security rules to incorporate changes. You can subscribe to an Amazon SNS topic and receive notifications when the IP address range file is updated. After you receive the notification, you can use code to retrieve the file, parse it, and make adjustments for your local environment. For more information, see Subscribe to AWS Public IP Address Changes by using Amazon SNS on the AWS News Blog.

Configure the security groups on your origin server to allow CloudFront traffic

If your origin uses Elastic Load Balancing, review the ELB security groups and make sure that the security groups allow inbound traffic from CloudFront.

You can also use AWS Lambda to automatically update your security groups to allow inbound traffic from CloudFront.

Make your custom origin server accessible on the internet

If CloudFront can't access your custom origin server because it isn't publicly available on the internet, CloudFront returns an HTTP 504 error.

CloudFront edge locations connect to origin servers through the internet. If your custom origin is on a private network, CloudFront can't reach it. Because of this, you can't use private servers, including internal Classic Load Balancers, as origin servers with CloudFront.

To check that internet traffic can connect to your origin server, run the following commands (where *`OriginDomainName`* is the domain name for your server):

For HTTPS traffic:

- nc -zv *`OriginDomainName`* 443
- telnet *`OriginDomainName`* 443

For HTTP traffic:

- nc -zv *`OriginDomainName`* 80

- telnet *OriginDomainName* 80

Find and fix delayed responses from applications on your origin server

Server timeouts are often the result of either an application taking a very long time to respond, or a timeout value that is set too low.

A quick fix to help avoid HTTP 504 errors is to simply set a higher CloudFront timeout value for your distribution. But we recommend that you first make sure that you address any performance and latency issues with the application and origin server. Then you can set a reasonable timeout value that helps prevent HTTP 504 errors and provides good responsiveness to users.

Here's an overview of the steps you can take to find performance issues and correct them:

1. Measure the typical and high-load latency (responsiveness) of your web application.
2. Add additional resources, such as CPU or memory, if needed. Take other steps to address issues, such as tuning database queries to accommodate high-load scenarios.
3. If needed, adjust the timeout value for your CloudFront distribution.

Following are details about each step.

Measure typical and high-load latency

To determine if one or more backend web application servers are experiencing high latency, run the following Linux curl command on each server:

```
curl -w "DNS Lookup Time: %{time_namelookup} \nConnect time: %{time_connect}
 \nTLS Setup: %{time_appconnect} \nRedirect Time: %{time_redirect} \nTime to first
 byte: %{time_starttransfer} \nTotal time: %{time_total} \n" -o /dev/null https://
www.example.com/yourobject
```

> **Note**
>
> If you run Windows on your servers, you can search for and download curl for Windows to run a similar command.

As you measure and evaluate the latency of an application that runs on your server, keep in mind the following:

- Latency values are relative to each application. However, a time to first byte in milliseconds rather than seconds or more, is reasonable.
- If you measure the application latency under normal load and it's fine, be aware that viewers might still experience timeouts under high load. When there is high demand, servers can have delayed responses or not respond at all. To help prevent high-load latency issues, check your server's resources such as CPU, memory, and disk reads and writes to make sure that your servers have the capacity to scale for high load.

  You can run the following Linux command to check the memory that is used by Apache processes:

  ```
  watch -n 1 "echo -n 'Apache Processes: ' && ps -C apache2 --no-headers | wc -l && free -m"
  ```
- High CPU utilization on the server can significantly reduce an application's performance. If you use an Amazon EC2 instance for your backend server, review the CloudWatch metrics for the server to check the CPU utilization. For more information, see the Amazon CloudWatch User Guide. Or if you're using your own server, refer to the server help documentation for instructions on how to check CPU utilization.
- Check for other potential issues under high loads, such as database queries that run slowly when there's a high volume of requests.

Add resources, and tune servers and databases

After you evaluate the responsiveness of your applications and servers, make sure that you have sufficient resources in place for typical traffic and high load situations:

- If you have your own server, make sure it has enough CPU, memory, and disk space to handle viewer requests, based on your evaluation.
- If you use an Amazon EC2 instance as your backend server, make sure that the instance type has the appropriate resources to fulfill incoming requests. For more information, see Instance types in the *Amazon EC2 User Guide*.

In addition, consider the following tuning steps to help avoid timeouts:

- If the Time to First Byte value that is returned by the curl command seems high, take steps to improve the performance of your application. Improving application responsiveness will in turn help reduce timeout errors.

- Tune database queries to make sure that they can handle high request volumes without slow performance.
- Set up keep-alive (persistent) connections on your backend server. This option helps to avoid latencies that occur when connections must be re-established for subsequent requests or users.
- **If you use Elastic Load Balancing as your origin**, the following are possible causes for a 504 error:
  - The load balancer fails to establish a connection to the target before the connection timeout expires (10 seconds).
  - The load balancer establishes a connection to the target but the target doesn't respond before the idle timeout period elapses.
  - The network access control list (ACL) for the subnet doesn't allow traffic from the targets to the load balancer nodes on the ephemeral ports (1024-65535).
  - The target returns a content-length header that is larger than the entity body. The load balancer times out waiting for the missing bytes.
  - The target is a Lambda function and Lambda doesn't respond before the connection timeout expires.

  For more information about reducing latency, see How do I troubleshoot high latency on my ELB Classic Load Balancer?
- **If you use MediaTailor as your origin**, the following are possible causes for a 504 error:
  - If relative URLs are mishandled, MediaTailor can receive malformed URLs from the players.
  - If MediaPackage is the manifest origin for MediaTailor, MediaPackage 404 manifest errors can cause MediaTailor to return a 504 error.
  - The request to the MediaTailor origin server takes more than 2 seconds to complete.
- **If you use Amazon API Gateway as your origin**, the following is a possible cause for a 504 error:
  - An integration request takes longer than your API Gateway REST API maximum integration timeout parameter. For more information, see How can I troubleshoot API HTTP 504 timeout errors with API Gateway?

If needed, adjust the CloudFront timeout value

You might have evaluated and addressed slow application performance, origin server capacity, and other issues, but viewers are still experiencing HTTP 504 errors. In this case, consider changing the

time that is specified in your distribution for origin response timeout. For more information, see the section called “Response timeout”.

Load testing CloudFront

Traditional load testing methods don't work well with CloudFront because CloudFront uses DNS to balance loads across geographically dispersed edge locations and within each edge location. When a client requests content from CloudFront, the client receives a DNS response that includes a set of IP addresses. If you test by sending requests to just one of the IP addresses that DNS returns, you're testing only a small subset of the resources in one CloudFront edge location, which doesn't accurately represent actual traffic patterns. Depending on the volume of data requested, testing in this way might overload and degrade the performance of that small subset of CloudFront servers.

CloudFront is designed to scale for viewers that have different client IP addresses and different DNS resolvers across multiple geographic regions. To perform load testing that accurately assesses CloudFront performance, we recommend that you do all of the following:

- Send client requests from multiple geographic regions.
- Configure your test so each client makes an independent DNS request. Each client will then receive a different set of IP addresses from DNS.
- For each client that is making requests, spread your client requests across the set of IP addresses that are returned by DNS. This makes sure that the load is distributed across multiple servers in a CloudFront edge location.

Notes

- Load testing isn't allowed on cache behaviors that have Lambda@Edge viewer request or viewer response triggers.
- Load testing isn't allowed on origins that have Origin Shield enabled.

Quotas

You can request a CloudFront quota increase by using the following options:

- You can use the Service Quotas console or the AWS Command Line Interface. For more information, see the following topics:
  - Requesting a quota increase in the *Service Quotas User Guide*
  - request-service-quota-increase in the *AWS CLI Command Reference*
- If a CloudFront quota isn't available in Service Quotas, use the AWS Support Center Console to create a service quota increase case.

CloudFront is subject to the following quotas.

Topics

- General quotas
- General quotas on distributions
- General quotas on policies
- Quotas on mTLS and trust stores
- Quotas on CloudFront Functions
- Quotas on CloudFront Functions network calls
- Quotas on Connection Functions
- Quotas on key value stores
- Quotas on Lambda@Edge
- Quotas on SSL certificates
- Quotas on invalidations
- Quotas on key groups
- Quotas on WebSocket connections
- Quotas on field-level encryption
- Quotas on cookies (legacy cache settings)
- Quotas on query strings (legacy cache settings)
- Quotas on headers
- Quotas on multi-tenant distributions

- Related information

General quotas

Entity	Default quota
Data transfer rate per distribution (This quota doesn't apply for distributions subscribed to CloudFront flat-rate pricing plans. For more information, see ???.)	150 Gbps Request a higher quota
Requests per second per distribution (This quota doesn't apply for distributions subscribed to CloudFront flat-rate pricing plans. For more information, see ???.)	250,000 Request a higher quota
Tags that can be added to a distribution	50 Request a higher quota
Files that you can serve per distribution	No quota
Maximum length of a request or an origin response, including headers and query strings, but not including the body content	32,768 bytes
Maximum length of a URL	8,192 bytes
Maximum number of real-time access log delivery configurations per AWS account	150
Maximum number of associations per web ACL	100 Request a higher quota

General quotas on distributions

Entity	Default quota
Alternate domain names (CNAMEs) per distribution For more information, see Use custom URLs by adding alternate domain names (CNAMEs).	100 Request a higher quota
Cache behaviors per distribution	75 Request a higher quota
Connection attempts per origin For more information, see Connection attempts.	1-3
Connection timeout per origin For more information, see Connection timeout.	1-10 seconds
Response timeout per origin This is also known as *origin read timeout* or *origin request timeout*. For more information, see Response timeout.	1-120 seconds Request a higher quota
Keep-alive timeout per origin For more information, see Keep-alive timeout (custom and VPC origins only).	1-300 seconds Request a higher quota
Distributions per AWS account For more information, see Create a distribution.	500 Request a higher quota
Distributions per origin access control	100 Request a higher quota

Entity	Default quota
Distributions within chain of requests to origin endpoint We don't recommend placing one distribution in front of another. E xceeding this quota results in a 403 error.	2
File compression: range of file sizes that CloudFront compresses For more information, see Serve compressed files.	1,000 to 10,000,000 bytes
Maximum cacheable file size per HTTP GET response. Only the responses for an HTTP GET are cached. Responses for POST or PUT are not cached.	50 GB
Maximum size of an HTTP request body.	64 GB
Origin access controls per AWS account	100 Request a higher quota
Origin access identities per AWS account	100 Request a higher quota
Origins per distribution	100 Request a higher quota
Origin groups per distribution	10 Request a higher quota

Entity	Default quota
Staging distributions per AWS account For more information, see the section called “Use continuous deployment to safely test changes”.	20 Request a higher quota
Distributions associated with the same VPC origin	50
VPC origins per AWS account	25 Request a higher quota
Maximum number of distributions that can be associated with a single Anycast static IP list.	100 Request a higher quota

General quotas on policies

Entity	Default quota
Custom cache policies per AWS account (Does not apply to CloudFront managed cache policies)	20 Request a higher quota
Distributions associated with the same cache policy	100
Query strings per cache policy	10 Request a higher quota
Headers per cache policy	10 Request a higher quota

Entity	Default quota
Cookies per cache policy	10 Request a higher quota
Total combined length of all query string, header, and cookie names in a cache policy	1024
Custom origin request policies per AWS account (Does not apply to CloudFront managed origin request policies)	20 Request a higher quota
Distributions associated with the same origin request policy	100
Query strings per origin request policy	10 Request a higher quota
Headers per origin request policy	10 Request a higher quota
Cookies per origin request policy	10 Request a higher quota
Total combined length of all query string, header, and cookie names in an origin request policy	1024
Custom response headers policies per AWS account (Does not apply to CloudFront managed response headers policies)	20 Request a higher quota

Entity	Default quota
Distributions associated with the same response headers policy	100 Request a higher quota
Custom headers per response headers policy	10 Request a higher quota
Continuous deployment policies per AWS account	20 Request a higher quota

Quotas on mTLS and trust stores

Entity	Default quota
Trust stores per AWS account	20 Request a higher quota
Distributions per trust store	300
CA bundle size	1 MB
Certificate size in CA bundle	16384 Request a higher quota
Number of certificates in CA bundle	25
Certificate chain depth	4

Quotas on CloudFront Functions

Entity	Default quota
Functions per AWS account	100
Maximum function size This quota isn't adjustable. To store additional data for your Cloud Front Functions, create a key value store and add your key-value pairs. For more information, see Amazon CloudFront KeyValueStore.	10 KB
Maximum function memory	2 MB
Distributions associated with the same function	100

In addition to these quotas, there are some other restrictions when using CloudFront Functions. For more information, see Restrictions on CloudFront Functions.

Quotas on CloudFront Functions network calls

Entity	Default quota
Network calls per function invocation (`cf.fetch()` and `cf.beacon()` combined)	1
Network call timeout, from sending the request to receiving the last byte	1 second
Maximum network call request body size	10 KB
Maximum network call response body size	10 KB
Maximum length of a network call request header name	256 bytes
Maximum length of a network call request header value	1,783 bytes
Maximum total size of all network call headers (request or response)	10 KB

Entity	Default quota
Maximum length of the network call path and query string	8 KB

These quotas aren't adjustable.

In addition to these quotas, there are some other restrictions when using CloudFront Functions. For more information, see Restrictions on CloudFront Functions.

Quotas on Connection Functions

Entity	Default quota
Connection Functions per AWS account	100
Maximum Connection Function size This quota isn't adjustable. To store additional data for your Connection Functions, create a key value store and add your key-value pairs. For more information, see Amazon CloudFront KeyValueStore.	10 KB
Maximum Connection Function memory	2 MB
Distributions associated with the same Connection Function	100

In addition to these quotas, there are some other restrictions when using Connection Functions. For more information, see Associate a CloudFront Connection Function.

Quotas on key value stores

Entity	Default quota
Maximum size of a key in a key-value pair	512 Bytes
Maximum size of the value in a key-value pair	1 KB

Entity	Default quota
Maximum key values pairs that you can update in a single API request	50 keys or 3 MB payload, whichever is reached first
Maximum size of an individual key value store	5 MB
Maximum number of functions that a single key value store can be associated with	100
Maximum number of key value stores per function	1
Maximum number of key value stores per account	200 Request a higher quota

Quotas on Lambda@Edge

General quotas

Entity	Default quota
Distributions per AWS account that can have Lambda@Edge functions	500 Request a higher quota
Lambda@Edge functions per distribution	100 Request a higher quota
Concurrent executions **Notes** •	1,000 (in each AWS Region) Request a higher quota

Entity	Default quota
AWS Lambda manages the concurrency quotas for Lambda@Edge. All Lambda functions in the AWS Region share this quota. • We recommend that you review the concurrent executions quota in all AWS Regions where you expect your viewer requests to come from. In addition, each instance of your Lambda@Edge function can serve up to 10 requests per second. The total invocation limit is 10 times your concurrency limit. For more information, see the following topics in the *AWS Lambda Developer Guide*: • Understanding Lambda function scaling • Lambda API requests	
Distributions associated with the same function	500
Maximum compressed size of a Lambda function and any included libraries	50 MB
Lambda@Edge requests per second (each supported AWS Region). For more information, see Concurrency quotas in the *AWS Lambda Developer Guide*.	10,000
Function memory size	Same as Lambda quotas
Function timeout. The function can make network calls to resources such as Amazon S3 buckets, DynamoDB tables, or Amazon EC2 instances in AWS Regions.	30 seconds

Quotas that differ by event type

Entity	Viewer request and viewer response events	Origin request and origin response events
Size of a response that is generated by a Lambda function, including headers and body	40 KB	1 MB

Notes

- For a list of additional Lambda@Edge quotas that can be increased from Service Quotas, see Amazon CloudFront endpoints and quotas in the *AWS General Reference*.
- In addition to these quotas, there are some other restrictions when using Lambda@Edge functions. For more information, see Restrictions on Lambda@Edge.

Quotas on SSL certificates

Entity	Default quota
SSL certificates per AWS account when serving HTTPS requests using dedicated IP addresses (no quota when serving HTTPS requests using SNI) For more information, see Use HTTPS with CloudFront.	2 Request a higher quota
SSL certificates that can be associated with a CloudFront distribution	1

If your SSL certificate is specifically for HTTPS communication between viewers and CloudFront, and if you used AWS Certificate Manager (ACM) or the IAM certificate store to provision or import your certificate, additional quotas apply. For more information, see Quotas on using SSL/TLS certificates with CloudFront (HTTPS between viewers and CloudFront only).

There are also quotas on the number of SSL certificates that you can import into AWS Certificate Manager (ACM) or upload to AWS Identity and Access Management (IAM). For more information, see Increase the quotas for SSL/TLS certificates.

Quotas on invalidations

Entity	Default quota
Number of paths or tags For more information, see Invalidate files to remove content and Invalidating content by cache tags.	150 paths or tags per second
Number of wildcard invalidations	1 wildcard invalidation per second
Tag invalidation: maximum number of files that one tag based invalidation can process	No quota
Wildcard invalidation: maximum number of files that one wildcard invalidation can process	No quota

Quotas on key groups

Entity	Default quota
Public keys in a single key group	5 Request a higher quota
Key groups associated with a single cache behavior	4 Request a higher quota
Key groups per AWS account	10

Entity	Default quota
	Request a higher quota
Distributions associated with a single key group	100 Request a higher quota

Quotas on WebSocket connections

Entity	Default quota
Origin response timeout (idle timeout)	10 minutes If CloudFront hasn't detected any bytes sent from the origin to the client within the past 10 minutes, the connection is assumed to be idle and is closed.

Quotas on field-level encryption

Entity	Default quota
Maximum length of a field to encrypt For more information, see Use field-level encryption to help protect sensitive data.	16 KB
Maximum number of fields in a request body when field-level encryption is configured	10

Entity	Default quota
Maximum length of a request body when field-level encryption is configured	1 MB
Maximum number of field-level encryption configurations that can be associated with one AWS account	10
Maximum number of field-level encryption profiles that can be associated with one AWS account	10
Maximum number of public keys that can be added to one AWS account	10
Maximum number of fields to encrypt that can be specified in one profile	10
Maximum number of CloudFront distributions that can be associated with a field-level encryption configuration	20
Maximum number of query argument profile mappings that can be included in a field-level encryption configuration	5

Quotas on cookies (legacy cache settings)

These quotas apply to CloudFront legacy cache settings. We recommend using a cache policy or origin request policy instead of the legacy settings.

Entity	Default quota
Cookies per cache behavior For more information, see Cache content based on cookies.	10 Request a higher quota
Total number of bytes in cookie names (doesn't apply if you configure CloudFront to forward all cookies to the origin)	512 minus the number of cookies

Quotas on query strings (legacy cache settings)

These quotas apply to CloudFront legacy cache settings. We recommend using a cache policy or origin request policy instead of the legacy settings.

Entity	Default quota
Maximum number of characters in a query string	128 characters
Maximum number of characters total for all query strings in the same parameter	512 characters
Query strings per cache behavior For more information, see Cache content based on query string parameters.	10 Request a higher quota

Quotas on headers

Entity	Default quota
Headers per cache behavior (legacy cache settings) For more information, see the section called “Cache content based on request headers”.	10 Request a higher quota
Forward headers per cache behavior	25 Request a higher quota
Custom headers: maximum number of custom headers that you can configure CloudFront to add to origin requests For more information, see the section called “Add custom headers to origin requests”.	30 Request a higher quota

Entity	Default quota
Custom headers: maximum number of custom headers that you can add to a response headers policy	10 Request a higher quota
Custom headers: maximum length of a header name	256 characters
Custom headers: maximum length of a header value	1,783 characters
Custom headers: maximum length of all header values and names combined	10,240 characters
Maximum length of the `Content-Security-Policy` header value	1,783 characters Request a higher quota
Maximum length of a `CORS (Access-Control-Allow-Origin)` header value	1,783 characters

Quotas on multi-tenant distributions

Entity	Default quota
Maximum number of distribution tenants per AWS account	1,000,000 Request a higher quota
Maximum number of multi-tenant distributions per AWS account	500 Request a higher quota
Maximum number of connection groups per AWS account	100

Entity	Default quota
	Request a higher quota
Maximum number of aliases per distribution tenant	100 Request a higher quota
Maximum number of parameters per distribution tenant	5 Request a higher quota
Maximum number of parameters per multi-tenant distribution	5 Request a higher quota
Maximum number of parameters in a field in a multi-tenant distribution	2 Request a higher quota
Maximum number of connection groups per Anycast static IP list	5 Request a higher quota

For more information about multi-tenant distributions, see Understand how multi-tenant distributions work.

Related information

For more information, see Amazon CloudFront endpoints and quotas in the *AWS General Reference*.

Code examples for CloudFront using AWS SDKs

The following code examples show how to use CloudFront with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Basic examples for CloudFront using AWS SDKs
  - Actions for CloudFront using AWS SDKs
    - Use CreateDistribution with an AWS SDK or CLI
    - Use CreateFunction with an AWS SDK
    - Use CreateInvalidation with a CLI
    - Use CreateKeyGroup with an AWS SDK
    - Use CreatePublicKey with an AWS SDK or CLI
    - Use DeleteDistribution with an AWS SDK or CLI
    - Use GetCloudFrontOriginAccessIdentity with a CLI
    - Use GetCloudFrontOriginAccessIdentityConfig with a CLI
    - Use GetDistribution with a CLI
    - Use GetDistributionConfig with an AWS SDK or CLI
    - Use ListCloudFrontOriginAccessIdentities with a CLI
    - Use ListDistributions with an AWS SDK or CLI
    - Use UpdateDistribution with an AWS SDK or CLI
- Scenarios for CloudFront using AWS SDKs
  - Create SaaS manager resources AWS SDK
  - Delete CloudFront signing resources using AWS SDK

- Get started with a basic content distribution network
- Getting started with web application firewalls
- Create signed URLs and cookies using an AWS SDK

- CloudFront Functions examples for CloudFront
  - Add HTTP security headers to a CloudFront Functions viewer response event
  - Add a CORS header to a CloudFront Functions viewer response event
  - Add a cache control header to a CloudFront Functions viewer response event
  - Add a true client IP header to a CloudFront Functions viewer request event
  - Add an origin header to a CloudFront Functions viewer request event
  - Add index.html to request URLs without a file name in a CloudFront Functions viewer request event
  - Authorize each request before the cache with a network call in a CloudFront Functions viewer request event
  - Authorize a request with a network call in a CloudFront Functions viewer request event
  - Check for bots with a network call that fails open in a CloudFront Functions viewer request event
  - Fail over to a secondary origin after a health check network call in a CloudFront Functions viewer request event
  - Read configuration through KeyValueStore with a network call in a CloudFront Functions viewer request event
  - Make a network call to an origin from a CloudFront Functions viewer request event
  - Serve markdown to AI agents and queue conversions with a network call from CloudFront Functions
  - Normalize query string parameters in a CloudFront Functions viewer request
  - Redirect to a new URL in a CloudFront Functions viewer request event
  - Rewrite a request URI based on KeyValueStore configuration for a CloudFront Functions viewer request event
  - Route a request through a private service behind a VPC origin with a network call in a CloudFront Functions viewer request event
  - Route requests to an origin closer to the viewer in a CloudFront Functions viewer request event
  - Send fire-and-forget events with a network call from CloudFront Functions
  - Assign a sticky A/B variant with KeyValueStore and a network call in CloudFront Functions

- Use key-value pairs in a CloudFront Functions viewer request
- Validate an Amazon Cognito access token with a network call in a CloudFront Functions viewer request event
- Validate a simple token in a CloudFront Functions viewer request
- Run a virtual waiting room with a network call in a CloudFront Functions viewer request event

Basic examples for CloudFront using AWS SDKs

The following code examples show how to use the basics of Amazon CloudFront with AWS SDKs.

Examples

- Actions for CloudFront using AWS SDKs
  - Use CreateDistribution with an AWS SDK or CLI
  - Use CreateFunction with an AWS SDK
  - Use CreateInvalidation with a CLI
  - Use CreateKeyGroup with an AWS SDK
  - Use CreatePublicKey with an AWS SDK or CLI
  - Use DeleteDistribution with an AWS SDK or CLI
  - Use GetCloudFrontOriginAccessIdentity with a CLI
  - Use GetCloudFrontOriginAccessIdentityConfig with a CLI
  - Use GetDistribution with a CLI
  - Use GetDistributionConfig with an AWS SDK or CLI
  - Use ListCloudFrontOriginAccessIdentities with a CLI
  - Use ListDistributions with an AWS SDK or CLI
  - Use UpdateDistribution with an AWS SDK or CLI

Actions for CloudFront using AWS SDKs

The following code examples demonstrate how to perform individual CloudFront actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the CloudFront API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for CloudFront using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon CloudFront API Reference.

Examples

- Use CreateDistribution with an AWS SDK or CLI
- Use CreateFunction with an AWS SDK
- Use CreateInvalidation with a CLI
- Use CreateKeyGroup with an AWS SDK
- Use CreatePublicKey with an AWS SDK or CLI
- Use DeleteDistribution with an AWS SDK or CLI
- Use GetCloudFrontOriginAccessIdentity with a CLI
- Use GetCloudFrontOriginAccessIdentityConfig with a CLI
- Use GetDistribution with a CLI
- Use GetDistributionConfig with an AWS SDK or CLI
- Use ListCloudFrontOriginAccessIdentities with a CLI
- Use ListDistributions with an AWS SDK or CLI
- Use UpdateDistribution with an AWS SDK or CLI

Use `CreateDistribution` with an AWS SDK or CLI

The following code examples show how to use `CreateDistribution`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Create a multi-tenant distribution and distribution tenant
- Get started with a basic content distribution network

CLI

AWS CLI

Example 1: To create a CloudFront distribution

The following `create-distribution` example creates a distribution for an S3 bucket named `amzn-s3-demo-bucket`, and also specifies `index.html` as the default root object, using command line arguments.

```
aws cloudfront create-distribution \
    --origin-domain-name amzn-s3-demo-bucket.s3.amazonaws.com \
    --default-root-object index.html
```

Output:

```
{
    "Location": "https://cloudfront.amazonaws.com/2019-03-26/distribution/
EMLARXS9EXAMPLE",
    "ETag": "E9LHASXEXAMPLE",
    "Distribution": {
        "Id": "EMLARXS9EXAMPLE",
        "ARN": "arn:aws:cloudfront::123456789012:distribution/EMLARXS9EXAMPLE",
        "Status": "InProgress",
        "LastModifiedTime": "2019-11-22T00:55:15.705Z",
        "InProgressInvalidationBatches": 0,
        "DomainName": "d111111abcdef8.cloudfront.net",
        "ActiveTrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "DistributionConfig": {
            "CallerReference": "cli-example",
            "Aliases": {
                "Quantity": 0
            },
            "DefaultRootObject": "index.html",
            "Origins": {
                "Quantity": 1,
                "Items": [
                    {
                        "Id": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-example",
                        "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                        "OriginPath": "",
                        "CustomHeaders": {
                            "Quantity": 0
                        },
                        "S3OriginConfig": {
                            "OriginAccessIdentity": ""
```

```
                        }
                    }
                ]
            },
            "OriginGroups": {
                "Quantity": 0
            },
            "DefaultCacheBehavior": {
                "TargetOriginId": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-
example",
                "ForwardedValues": {
                    "QueryString": false,
                    "Cookies": {
                        "Forward": "none"
                    },
                    "Headers": {
                        "Quantity": 0
                    },
                    "QueryStringCacheKeys": {
                        "Quantity": 0
                    }
                },
                "TrustedSigners": {
                    "Enabled": false,
                    "Quantity": 0
                },
                "ViewerProtocolPolicy": "allow-all",
                "MinTTL": 0,
                "AllowedMethods": {
                    "Quantity": 2,
                    "Items": [
                        "HEAD",
                        "GET"
                    ],
                    "CachedMethods": {
                        "Quantity": 2,
                        "Items": [
                            "HEAD",
                            "GET"
                        ]
                    }
                },
                "SmoothStreaming": false,
                "DefaultTTL": 86400,
```

```
                "MaxTTL": 31536000,
                "Compress": false,
                "LambdaFunctionAssociations": {
                    "Quantity": 0
                },
                "FieldLevelEncryptionId": ""
            },
            "CacheBehaviors": {
                "Quantity": 0
            },
            "CustomErrorResponses": {
                "Quantity": 0
            },
            "Comment": "",
            "Logging": {
                "Enabled": false,
                "IncludeCookies": false,
                "Bucket": "",
                "Prefix": ""
            },
            "PriceClass": "PriceClass_All",
            "Enabled": true,
            "ViewerCertificate": {
                "CloudFrontDefaultCertificate": true,
                "MinimumProtocolVersion": "TLSv1",
                "CertificateSource": "cloudfront"
            },
            "Restrictions": {
                "GeoRestriction": {
                    "RestrictionType": "none",
                    "Quantity": 0
                }
            },
            "WebACLId": "",
            "HttpVersion": "http2",
            "IsIPV6Enabled": true
        }
    }
}
```

Example 2: To create a CloudFront distribution using a JSON file

The following `create-distribution` example creates a distribution for an S3 bucket named `amzn-s3-demo-bucket`, and also specifies `index.html` as the default root object, using a JSON file.

```
aws cloudfront create-distribution \
    --distribution-config file://dist-config.json
```

Contents of `dist-config.json`:

```
{
    "CallerReference": "cli-example",
    "Aliases": {
        "Quantity": 0
    },
    "DefaultRootObject": "index.html",
    "Origins": {
        "Quantity": 1,
        "Items": [
            {
                "Id": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-example",
                "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                "OriginPath": "",
                "CustomHeaders": {
                    "Quantity": 0
                },
                "S3OriginConfig": {
                    "OriginAccessIdentity": ""
                }
            }
        ]
    },
    "OriginGroups": {
        "Quantity": 0
    },
    "DefaultCacheBehavior": {
        "TargetOriginId": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-example",
        "ForwardedValues": {
            "QueryString": false,
            "Cookies": {
                "Forward": "none"
            },
            "Headers": {
                "Quantity": 0
```

```
            },
            "QueryStringCacheKeys": {
                "Quantity": 0
            }
        },
        "TrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "ViewerProtocolPolicy": "allow-all",
        "MinTTL": 0,
        "AllowedMethods": {
            "Quantity": 2,
            "Items": [
                "HEAD",
                "GET"
            ],
            "CachedMethods": {
                "Quantity": 2,
                "Items": [
                    "HEAD",
                    "GET"
                ]
            }
        },
        "SmoothStreaming": false,
        "DefaultTTL": 86400,
        "MaxTTL": 31536000,
        "Compress": false,
        "LambdaFunctionAssociations": {
            "Quantity": 0
        },
        "FieldLevelEncryptionId": ""
    },
    "CacheBehaviors": {
        "Quantity": 0
    },
    "CustomErrorResponses": {
        "Quantity": 0
    },
    "Comment": "",
    "Logging": {
        "Enabled": false,
        "IncludeCookies": false,
```

```
        "Bucket": "",
        "Prefix": ""
    },
    "PriceClass": "PriceClass_All",
    "Enabled": true,
    "ViewerCertificate": {
        "CloudFrontDefaultCertificate": true,
        "MinimumProtocolVersion": "TLSv1",
        "CertificateSource": "cloudfront"
    },
    "Restrictions": {
        "GeoRestriction": {
            "RestrictionType": "none",
            "Quantity": 0
        }
    },
    "WebACLId": "",
    "HttpVersion": "http2",
    "IsIPV6Enabled": true
}
```

Output:

```
{
    "Location": "https://cloudfront.amazonaws.com/2019-03-26/distribution/
EMLARXS9EXAMPLE",
    "ETag": "E9LHASXEXAMPLE",
    "Distribution": {
        "Id": "EMLARXS9EXAMPLE",
        "ARN": "arn:aws:cloudfront::123456789012:distribution/EMLARXS9EXAMPLE",
        "Status": "InProgress",
        "LastModifiedTime": "2019-11-22T00:55:15.705Z",
        "InProgressInvalidationBatches": 0,
        "DomainName": "d111111abcdef8.cloudfront.net",
        "ActiveTrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "DistributionConfig": {
            "CallerReference": "cli-example",
            "Aliases": {
                "Quantity": 0
            },
```

```
            "DefaultRootObject": "index.html",
            "Origins": {
                "Quantity": 1,
                "Items": [
                    {
                        "Id": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-example",
                        "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                        "OriginPath": "",
                        "CustomHeaders": {
                            "Quantity": 0
                        },
                        "S3OriginConfig": {
                            "OriginAccessIdentity": ""
                        }
                    }
                ]
            },
            "OriginGroups": {
                "Quantity": 0
            },
            "DefaultCacheBehavior": {
                "TargetOriginId": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-
example",
                "ForwardedValues": {
                    "QueryString": false,
                    "Cookies": {
                        "Forward": "none"
                    },
                    "Headers": {
                        "Quantity": 0
                    },
                    "QueryStringCacheKeys": {
                        "Quantity": 0
                    }
                },
                "TrustedSigners": {
                    "Enabled": false,
                    "Quantity": 0
                },
                "ViewerProtocolPolicy": "allow-all",
                "MinTTL": 0,
                "AllowedMethods": {
                    "Quantity": 2,
                    "Items": [
```

```
                    "HEAD",
                    "GET"
                ],
                "CachedMethods": {
                    "Quantity": 2,
                    "Items": [
                        "HEAD",
                        "GET"
                    ]
                }
            },
            "SmoothStreaming": false,
            "DefaultTTL": 86400,
            "MaxTTL": 31536000,
            "Compress": false,
            "LambdaFunctionAssociations": {
                "Quantity": 0
            },
            "FieldLevelEncryptionId": ""
        },
        "CacheBehaviors": {
            "Quantity": 0
        },
        "CustomErrorResponses": {
            "Quantity": 0
        },
        "Comment": "",
        "Logging": {
            "Enabled": false,
            "IncludeCookies": false,
            "Bucket": "",
            "Prefix": ""
        },
        "PriceClass": "PriceClass_All",
        "Enabled": true,
        "ViewerCertificate": {
            "CloudFrontDefaultCertificate": true,
            "MinimumProtocolVersion": "TLSv1",
            "CertificateSource": "cloudfront"
        },
        "Restrictions": {
            "GeoRestriction": {
                "RestrictionType": "none",
                "Quantity": 0
```

```
                }
            },
            "WebACLId": "",
            "HttpVersion": "http2",
            "IsIPV6Enabled": true
        }
    }
}
```

Example 3: To create a CloudFront multi-tenant distribution with a certificate

The following `create-distribution` example creates a CloudFront distribution with multi-tenant support and a specifies a TLS certificate.

```
aws cloudfront create-distribution \
    --distribution-config file://dist-config.json
```

Contents of `dist-config.json`:

```
{
    "CallerReference": "cli-example-with-cert",
    "Comment": "CLI example distribution",
    "DefaultRootObject": "index.html",
    "Origins": {
        "Quantity": 1,
        "Items": [
            {
                "Id": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
                "DomainName": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
                "OriginPath": "/{{tenantName}}",
                "CustomHeaders": {
                    "Quantity": 0
                },
                "S3OriginConfig": {
                    "OriginAccessIdentity": ""
                }
            }
        ]
    },
    "DefaultCacheBehavior": {
        "TargetOriginId": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
        "CachePolicyId": "658327ea-f89d-4fab-a63d-7e88639e5ABC",
        "ViewerProtocolPolicy": "allow-all",
```

```
        "AllowedMethods": {
            "Quantity": 2,
            "Items": ["HEAD", "GET"],
            "CachedMethods": {
                "Quantity": 2,
                "Items": ["HEAD", "GET"]
            }
        }
    },
    "Enabled": true,
    "ViewerCertificate": {
        "ACMCertificateArn": "arn:aws:acm:us-east-1:123456789012:certificate/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSLSupportMethod": "sni-only"
    },
    "HttpVersion": "http2",
    "ConnectionMode": "tenant-only",
    "TenantConfig": {
        "ParameterDefinitions": [
            {
                "Name": "tenantName",
                "Definition": {
                    "StringSchema": {
                        "Comment": "tenantName parameter",
                        "DefaultValue": "root",
                        "Required": false
                    }
                }
            }
        ]
    }
}
```

Output:

```
{
    "Location": "https://cloudfront.amazonaws.com/2020-05-31/distribution/
E1HVIAU7UABC",
    "ETag": "E20LT7R1BABC",
    "Distribution": {
        "Id": "E1HVIAU7U12ABC",
        "ARN": "arn:aws:cloudfront::123456789012:distribution/E1HVIAU7U12ABC",
        "Status": "InProgress",
```

```
        "LastModifiedTime": "2025-07-10T20:33:31.117000+00:00",
        "InProgressInvalidationBatches": 0,
        "DomainName": "example.com",
        "ActiveTrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "ActiveTrustedKeyGroups": {
            "Enabled": false,
            "Quantity": 0
        },
        "DistributionConfig": {
            "CallerReference": "cli-example-with-cert",
            "DefaultRootObject": "index.html",
            "Origins": {
                "Quantity": 1,
                "Items": [
                    {
                        "Id": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
                        "DomainName": "amzn-s3-demo-bucket.s3.us-
west-2.amazonaws.com",
                        "OriginPath": "/{{tenantName}}",
                        "CustomHeaders": {
                            "Quantity": 0
                        },
                        "S3OriginConfig": {
                            "OriginAccessIdentity": ""
                        },
                        "ConnectionAttempts": 3,
                        "ConnectionTimeout": 10,
                        "OriginShield": {
                            "Enabled": false
                        },
                        "OriginAccessControlId": ""
                    }
                ]
            },
            "OriginGroups": {
                "Quantity": 0
            },
            "DefaultCacheBehavior": {
                "TargetOriginId": "amzn-s3-demo-bucket.s3.us-
west-2.amazonaws.com",
                "TrustedKeyGroups": {
```

```
                    "Enabled": false,
                    "Quantity": 0
                },
                "ViewerProtocolPolicy": "allow-all",
                "AllowedMethods": {
                    "Quantity": 2,
                    "Items": ["HEAD", "GET"],
                    "CachedMethods": {
                        "Quantity": 2,
                        "Items": ["HEAD", "GET"]
                    }
                },
                "Compress": false,
                "LambdaFunctionAssociations": {
                    "Quantity": 0
                },
                "FunctionAssociations": {
                    "Quantity": 0
                },
                "FieldLevelEncryptionId": "",
                "CachePolicyId": "658327ea-f89d-4fab-a63d-7e88639e5ABC",
                "GrpcConfig": {
                    "Enabled": false
                }
            },
            "CacheBehaviors": {
                "Quantity": 0
            },
            "CustomErrorResponses": {
                "Quantity": 0
            },
            "Comment": "CLI example distribution",
            "Logging": {
                "Enabled": false,
                "IncludeCookies": false,
                "Bucket": "",
                "Prefix": ""
            },
            "Enabled": true,
            "ViewerCertificate": {
                "CloudFrontDefaultCertificate": false,
                "ACMCertificateArn": "arn:aws:acm:us-
east-1:123456789012:certificate/1954f095-11b6-4daf-9952-0c308a00abc",
                "SSLSupportMethod": "sni-only",
```

```
                "MinimumProtocolVersion": "TLSv1.2_2021",
                "Certificate": "arn:aws:acm:us-
east-1:123456789012:certificate/1954f095-11b6-4daf-9952-0c308a00abc",
                "CertificateSource": "acm"
            },
            "Restrictions": {
                "GeoRestriction": {
                    "RestrictionType": "none",
                    "Quantity": 0
                }
            },
            "WebACLId": "",
            "HttpVersion": "http2",
            "TenantConfig": {
                "ParameterDefinitions": [
                    {
                        "Name": "tenantName",
                        "Definition": {
                            "StringSchema": {
                                "Comment": "tenantName parameter",
                                "DefaultValue": "root",
                                "Required": false
                            }
                        }
                    }
                ]
            },
            "ConnectionMode": "tenant-only"
        }
    }
}
```

For more information, see Working with distributions in the *Amazon CloudFront Developer Guide*.

Example 4: To create a CloudFront multi-tenant distribution without a certificate

The following `create-distribution` example creates a CloudFront distribution with multi-tenant support but without a TLS certificate.

```
aws cloudfront create-distribution \
    --distribution-config file://dist-config.json
```

Contents of `dist-config.json`:

```
{
    "CallerReference": "cli-example",
    "Comment": "CLI example distribution",
    "DefaultRootObject": "index.html",
    "Origins": {
        "Quantity": 1,
        "Items": [
            {
                "Id": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
                "DomainName": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
                "OriginPath": "/{{tenantName}}",
                "CustomHeaders": {
                    "Quantity": 0
                },
                "S3OriginConfig": {
                    "OriginAccessIdentity": ""
                }
            }
        ]
    },
    "DefaultCacheBehavior": {
        "TargetOriginId": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
        "CachePolicyId": "658327ea-f89d-4fab-a63d-7e88639e5ABC",
        "ViewerProtocolPolicy": "allow-all",
        "AllowedMethods": {
            "Quantity": 2,
            "Items": [
                "HEAD",
                "GET"
            ],
            "CachedMethods": {
                "Quantity": 2,
                "Items": [
                    "HEAD",
                    "GET"
                ]
            }
        }
    },
    "Enabled": true,
    "HttpVersion": "http2",
    "ConnectionMode": "tenant-only",
```

```
    "TenantConfig": {
        "ParameterDefinitions": [
            {
                "Name": "tenantName",
                "Definition": {
                    "StringSchema": {
                        "Comment": "tenantName parameter",
                        "DefaultValue": "root",
                        "Required": false
                    }
                }
            }
        ]
    }
}
```

Output:

```
{
    "Location": "https://cloudfront.amazonaws.com/2020-05-31/distribution/
E2GJ5J9QN12ABC",
    "ETag": "E37YLVVQIABC",
    "Distribution": {
        "Id": "E2GJ5J9QNABC",
        "ARN": "arn:aws:cloudfront::123456789012:distribution/E2GJ5J9QN12ABC",
        "Status": "InProgress",
        "LastModifiedTime": "2025-07-10T20:35:20.565000+00:00",
        "InProgressInvalidationBatches": 0,
        "DomainName": "example.com",
        "ActiveTrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "ActiveTrustedKeyGroups": {
            "Enabled": false,
            "Quantity": 0
        },
        "DistributionConfig": {
            "CallerReference": "cli-example-no-cert",
            "DefaultRootObject": "index.html",
            "Origins": {
                "Quantity": 1,
                "Items": [
```

```
                    {
                        "Id": "amzn-s3-demo-bucket.s3.us-west-2.amazonaws.com",
                        "DomainName": "amzn-s3-demo-bucket.s3.us-
west-2.amazonaws.com",
                        "OriginPath": "/{{tenantName}}",
                        "CustomHeaders": {
                            "Quantity": 0
                        },
                        "S3OriginConfig": {
                            "OriginAccessIdentity": ""
                        },
                        "ConnectionAttempts": 3,
                        "ConnectionTimeout": 10,
                        "OriginShield": {
                            "Enabled": false
                        },
                        "OriginAccessControlId": ""
                    }
                ]
            },
            "OriginGroups": {
                "Quantity": 0
            },
            "DefaultCacheBehavior": {
                "TargetOriginId": "amzn-s3-demo-bucket.s3.us-
west-2.amazonaws.com",
                "TrustedKeyGroups": {
                    "Enabled": false,
                    "Quantity": 0
                },
                "ViewerProtocolPolicy": "allow-all",
                "AllowedMethods": {
                    "Quantity": 2,
                    "Items": [
                        "HEAD",
                        "GET"
                    ],
                    "CachedMethods": {
                        "Quantity": 2,
                        "Items": [
                            "HEAD",
                            "GET"
                        ]
                    }
```

```
                },
                "Compress": false,
                "LambdaFunctionAssociations": {
                    "Quantity": 0
                },
                "FunctionAssociations": {
                    "Quantity": 0
                },
                "FieldLevelEncryptionId": "",
                "CachePolicyId": "658327ea-f89d-4fab-a63d-7e88639e5ABC",
                "GrpcConfig": {
                    "Enabled": false
                }
            },
            "CacheBehaviors": {
                "Quantity": 0
            },
            "CustomErrorResponses": {
                "Quantity": 0
            },
            "Comment": "CLI example distribution",
            "Logging": {
                "Enabled": false,
                "IncludeCookies": false,
                "Bucket": "",
                "Prefix": ""
            },
            "Enabled": true,
            "ViewerCertificate": {
                "CloudFrontDefaultCertificate": true,
                "SSLSupportMethod": "sni-only",
                "MinimumProtocolVersion": "TLSv1",
                "CertificateSource": "cloudfront"
            },
            "Restrictions": {
                "GeoRestriction": {
                    "RestrictionType": "none",
                    "Quantity": 0
                }
            },
            "WebACLId": "",
            "HttpVersion": "http2",
            "TenantConfig": {
                "ParameterDefinitions": [
```

```
                    {
                        "Name": "tenantName",
                        "Definition": {
                            "StringSchema": {
                                "Comment": "tenantName parameter",
                                "DefaultValue": "root",
                                "Required": false
                            }
                        }
                    }
                ]
            },
            "ConnectionMode": "tenant-only"
        }
    }
}
```

For more information, see Configure distributions in the *Amazon CloudFront Developer Guide*.

- For API details, see CreateDistribution in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The following example uses an Amazon Simple Storage Service (Amazon S3) bucket as a content origin.

After creating the distribution, the code creates a CloudFrontWaiter to wait until the distribution is deployed before returning the distribution.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.internal.waiters.ResponseOrException;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
```

```
import
 software.amazon.awssdk.services.cloudfront.model.CreateDistributionResponse;
import software.amazon.awssdk.services.cloudfront.model.Distribution;
import software.amazon.awssdk.services.cloudfront.model.GetDistributionResponse;
import software.amazon.awssdk.services.cloudfront.model.ItemSelection;
import software.amazon.awssdk.services.cloudfront.model.Method;
import software.amazon.awssdk.services.cloudfront.model.ViewerProtocolPolicy;
import software.amazon.awssdk.services.cloudfront.waiters.CloudFrontWaiter;
import software.amazon.awssdk.services.s3.S3Client;

import java.time.Instant;

public class CreateDistribution {

        private static final Logger logger =
 LoggerFactory.getLogger(CreateDistribution.class);

        public static Distribution createDistribution(CloudFrontClient
 cloudFrontClient, S3Client s3Client,
                        final String bucketName, final String keyGroupId, final
 String originAccessControlId) {

                final String region = s3Client.headBucket(b ->
 b.bucket(bucketName)).sdkHttpResponse().headers()
                                .get("x-amz-bucket-region").get(0);
                final String originDomain = bucketName + ".s3." + region +
 ".amazonaws.com";
                String originId = originDomain; // Use the originDomain value for
 the originId.

                // The service API requires some deprecated methods, such as
                // DefaultCacheBehavior.Builder#minTTL and #forwardedValue.
                CreateDistributionResponse createDistResponse =
 cloudFrontClient.createDistribution(builder -> builder
                                .distributionConfig(b1 -> b1
                                                .origins(b2 -> b2
                                                                .quantity(1)
                                                                .items(b3 -> b3

 .domainName(originDomain)

 .id(originId)

 .s3OriginConfig(builder4 -> builder4
```

```
                .originAccessIdentity(
                                    ""))
.originAccessControlId(
                originAccessControlId)))
                                                .defaultCacheBehavior(b2 -> b2
.viewerProtocolPolicy(ViewerProtocolPolicy.ALLOW_ALL)
.targetOriginId(originId)
                                                                    .minTTL(200L)
.forwardedValues(b5 -> b5
.cookies(cp -> cp
                .forward(ItemSelection.NONE))
.queryString(true))
.trustedKeyGroups(b3 -> b3
.quantity(1)
.items(keyGroupId)
.enabled(true))
.allowedMethods(b4 -> b4
.quantity(2)
.items(Method.HEAD, Method.GET)
.cachedMethods(b5 -> b5
                .quantity(2)
                .items(Method.HEAD,
                                    Method.GET))))
```

```
                                                .cacheBehaviors(b -> b
                                                                .quantity(1)
                                                                .items(b2 -> b2

.pathPattern("/index.html")

.viewerProtocolPolicy(

              ViewerProtocolPolicy.ALLOW_ALL)

.targetOriginId(originId)

.trustedKeyGroups(b3 -> b3

              .quantity(1)

              .items(keyGroupId)

              .enabled(true))

.minTTL(200L)

.forwardedValues(b4 -> b4

              .cookies(cp -> cp

                              .forward(ItemSelection.NONE))

              .queryString(true))

.allowedMethods(b5 -> b5.quantity(2)

              .items(Method.HEAD,

                             Method.GET)

              .cachedMethods(b6 -> b6

                              .quantity(2)

                              .items(Method.HEAD,

                                                    Method.GET)))))
                                                     .enabled(true)
```

```
                                                .comment("Distribution built with
 java")

 .callerReference(Instant.now().toString())));

                final Distribution distribution =
 createDistResponse.distribution();
                logger.info("Distribution created. DomainName: [{}]  Id: [{}]",
 distribution.domainName(),
                                distribution.id());
                logger.info("Waiting for distribution to be deployed ...");
                try (CloudFrontWaiter cfWaiter =
 CloudFrontWaiter.builder().client(cloudFrontClient).build()) {
                        ResponseOrException<GetDistributionResponse>
 responseOrException = cfWaiter
                                        .waitUntilDistributionDeployed(builder ->
 builder.id(distribution.id()))
                                        .matched();
                        responseOrException.response()
                                        .orElseThrow(() -> new
 RuntimeException("Distribution not created"));
                        logger.info("Distribution deployed. DomainName: [{}]  Id:
 [{}]", distribution.domainName(),
                                        distribution.id());
                }
                return distribution;
        }
}
```

- For API details, see [CreateDistribution](#) in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Creates a basic CloudFront distribution, configured with logging and caching.

```
$origin = New-Object Amazon.CloudFront.Model.Origin
$origin.DomainName = "amzn-s3-demo-bucket.s3.amazonaws.com"
$origin.Id = "UniqueOrigin1"
$origin.S3OriginConfig = New-Object Amazon.CloudFront.Model.S3OriginConfig
$origin.S3OriginConfig.OriginAccessIdentity = ""
```

```
New-CFDistribution `
      -DistributionConfig_Enabled $true `
      -DistributionConfig_Comment "Test distribution" `
      -Origins_Item $origin `
      -Origins_Quantity 1 `
      -Logging_Enabled $true `
      -Logging_IncludeCookie $true `
      -Logging_Bucket amzn-s3-demo-logging-bucket.s3.amazonaws.com `
      -Logging_Prefix "help/" `
      -DistributionConfig_CallerReference Client1 `
      -DistributionConfig_DefaultRootObject index.html `
      -DefaultCacheBehavior_TargetOriginId $origin.Id `
      -ForwardedValues_QueryString $true `
      -Cookies_Forward all `
      -WhitelistedNames_Quantity 0 `
      -TrustedSigners_Enabled $false `
      -TrustedSigners_Quantity 0 `
      -DefaultCacheBehavior_ViewerProtocolPolicy allow-all `
      -DefaultCacheBehavior_MinTTL 1000 `
      -DistributionConfig_PriceClass "PriceClass_All" `
      -CacheBehaviors_Quantity 0 `
      -Aliases_Quantity 0
```

- For API details, see CreateDistribution in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Creates a basic CloudFront distribution, configured with logging and caching.

```
$origin = New-Object Amazon.CloudFront.Model.Origin
$origin.DomainName = "amzn-s3-demo-bucket.s3.amazonaws.com"
$origin.Id = "UniqueOrigin1"
$origin.S3OriginConfig = New-Object Amazon.CloudFront.Model.S3OriginConfig
$origin.S3OriginConfig.OriginAccessIdentity = ""
New-CFDistribution `
      -DistributionConfig_Enabled $true `
      -DistributionConfig_Comment "Test distribution" `
      -Origins_Item $origin `
      -Origins_Quantity 1 `
      -Logging_Enabled $true `
      -Logging_IncludeCookie $true `
      -Logging_Bucket amzn-s3-demo-logging-bucket.s3.amazonaws.com `
      -Logging_Prefix "help/" `
      -DistributionConfig_CallerReference Client1 `
```

```
    -DistributionConfig_DefaultRootObject index.html `
    -DefaultCacheBehavior_TargetOriginId $origin.Id `
    -ForwardedValues_QueryString $true `
    -Cookies_Forward all `
    -WhitelistedNames_Quantity 0 `
    -TrustedSigners_Enabled $false `
    -TrustedSigners_Quantity 0 `
    -DefaultCacheBehavior_ViewerProtocolPolicy allow-all `
    -DefaultCacheBehavior_MinTTL 1000 `
    -DistributionConfig_PriceClass "PriceClass_All" `
    -CacheBehaviors_Quantity 0 `
    -Aliases_Quantity 0
```

- For API details, see CreateDistribution in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateFunction` with an AWS SDK

The following code example shows how to use `CreateFunction`.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import software.amazon.awssdk.services.cloudfront.model.CloudFrontException;
import software.amazon.awssdk.services.cloudfront.model.CreateFunctionRequest;
import software.amazon.awssdk.services.cloudfront.model.CreateFunctionResponse;
import software.amazon.awssdk.services.cloudfront.model.FunctionConfig;
import software.amazon.awssdk.services.cloudfront.model.FunctionRuntime;
```

```
import java.io.InputStream;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateFunction {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <functionName> <filePath>

                Where:
                    functionName - The name of the function to create.\s
                    filePath - The path to a file that contains the application
 logic for the function.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String functionName = args[0];
        String filePath = args[1];
        CloudFrontClient cloudFrontClient = CloudFrontClient.builder()
                .region(Region.AWS_GLOBAL)
                .build();

        String funArn = createNewFunction(cloudFrontClient, functionName,
 filePath);
        System.out.println("The function ARN is " + funArn);
        cloudFrontClient.close();
    }

    public static String createNewFunction(CloudFrontClient cloudFrontClient,
 String functionName, String filePath) {
```

```
        try {
            InputStream fileIs =
 CreateFunction.class.getClassLoader().getResourceAsStream(filePath);
            SdkBytes functionCode = SdkBytes.fromInputStream(fileIs);

            FunctionConfig config = FunctionConfig.builder()
                    .comment("Created by using the CloudFront Java API")
                    .runtime(FunctionRuntime.CLOUDFRONT_JS_1_0)
                    .build();

            CreateFunctionRequest functionRequest =
 CreateFunctionRequest.builder()
                    .name(functionName)
                    .functionCode(functionCode)
                    .functionConfig(config)
                    .build();

            CreateFunctionResponse response =
 cloudFrontClient.createFunction(functionRequest);
            return response.functionSummary().functionMetadata().functionARN();

        } catch (CloudFrontException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see CreateFunction in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateInvalidation` with a CLI

The following code examples show how to use `CreateInvalidation`.

CLI

AWS CLI

To create an invalidation for a CloudFront distribution

The following `create-invalidation` example creates an invalidation for the specified files in the specified CloudFront distribution.

```
aws cloudfront create-invalidation \
    --distribution-id EDFDVBD6EXAMPLE \
    --paths "/example-path/example-file.jpg" "/example-path/example-file2.png"
```

Output:

```
{
    "Location": "https://cloudfront.amazonaws.com/2019-03-26/distribution/
EDFDVBD6EXAMPLE/invalidation/I1JLWSDAP8FU89",
    "Invalidation": {
        "Id": "I1JLWSDAP8FU89",
        "Status": "InProgress",
        "CreateTime": "2019-12-05T18:24:51.407Z",
        "InvalidationBatch": {
            "Paths": {
                "Quantity": 2,
                "Items": [
                    "/example-path/example-file2.png",
                    "/example-path/example-file.jpg"
                ]
            },
            "CallerReference": "cli-1575570291-670203"
        }
    }
}
```

In the previous example, the AWS CLI automatically generated a random `CallerReference`. To specify your own `CallerReference`, or to avoid passing the invalidation parameters as command line arguments, you can use a JSON file. The following example creates an invalidation for two files, by providing the invalidation parameters in a JSON file named `inv-batch.json`.

```
aws cloudfront create-invalidation \
```

```
    --distribution-id EDFDVBD6EXAMPLE \
    --invalidation-batch file://inv-batch.json
```

Contents of `inv-batch.json`:

```
{
    "Paths": {
        "Quantity": 2,
        "Items": [
            "/example-path/example-file.jpg",
            "/example-path/example-file2.png"
        ]
    },
    "CallerReference": "cli-example"
}
```

Output:

```
{
    "Location": "https://cloudfront.amazonaws.com/2019-03-26/distribution/
EDFDVBD6EXAMPLE/invalidation/I2J0I21PCUYOIK",
    "Invalidation": {
        "Id": "I2J0I21PCUYOIK",
        "Status": "InProgress",
        "CreateTime": "2019-12-05T18:40:49.413Z",
        "InvalidationBatch": {
            "Paths": {
                "Quantity": 2,
                "Items": [
                    "/example-path/example-file.jpg",
                    "/example-path/example-file2.png"
                ]
            },
            "CallerReference": "cli-example"
        }
    }
}
```

For more information, see Invalidate files to remove content in the *Amazon CloudFront Developer Guide*.

- For API details, see CreateInvalidation in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example creates a new invalidation on a distribution with an ID of EXAMPLENSTXAXE. The CallerReference is a unique ID chosen by the user; in this case, a time stamp representing May 15, 2019 at 9:00 a.m. is used. The $Paths variable stores three paths to image and media files that the user does not want as part of the distribution's cache. The -Paths_Quantity parameter value is the total number of paths specified in the -Paths_Item parameter.

```
$Paths = "/images/*.gif", "/images/image1.jpg", "/videos/*.mp4"
New-CFInvalidation -DistributionId "EXAMPLENSTXAXE" -
InvalidationBatch_CallerReference 20190515090000 -Paths_Item $Paths -
Paths_Quantity 3
```

Output:

```
Invalidation                          Location

------------                          --------

Amazon.CloudFront.Model.Invalidation https://cloudfront.amazonaws.com/2018-11-05/
distribution/EXAMPLENSTXAXE/invalidation/EXAMPLE8NOK9H
```

- For API details, see CreateInvalidation in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example creates a new invalidation on a distribution with an ID of EXAMPLENSTXAXE. The CallerReference is a unique ID chosen by the user; in this case, a time stamp representing May 15, 2019 at 9:00 a.m. is used. The $Paths variable stores three paths to image and media files that the user does not want as part of the distribution's cache. The -Paths_Quantity parameter value is the total number of paths specified in the -Paths_Item parameter.

```
$Paths = "/images/*.gif", "/images/image1.jpg", "/videos/*.mp4"
New-CFInvalidation -DistributionId "EXAMPLENSTXAXE" -
InvalidationBatch_CallerReference 20190515090000 -Paths_Item $Paths -
Paths_Quantity 3
```

Output:

```
Invalidation                          Location

------------                          --------

Amazon.CloudFront.Model.Invalidation https://cloudfront.amazonaws.com/2018-11-05/
distribution/EXAMPLENSTXAXE/invalidation/EXAMPLE8NOK9H
```

- For API details, see CreateInvalidation in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateKeyGroup` with an AWS SDK

The following code example shows how to use `CreateKeyGroup`.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

A key group requires at least one public key that is used to verify signed URLs or cookies.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;

import java.util.UUID;

public class CreateKeyGroup {
    private static final Logger logger =
 LoggerFactory.getLogger(CreateKeyGroup.class);
```

```
    public static String createKeyGroup(CloudFrontClient cloudFrontClient, String
 publicKeyId) {
        String keyGroupId = cloudFrontClient.createKeyGroup(b ->
 b.keyGroupConfig(c -> c
                .items(publicKeyId)
                .name("JavaKeyGroup" + UUID.randomUUID())))
                .keyGroup().id();
        logger.info("KeyGroup created with ID: [{}]", keyGroupId);
        return keyGroupId;
    }
}
```

- For API details, see CreateKeyGroup in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreatePublicKey` with an AWS SDK or CLI

The following code examples show how to use `CreatePublicKey`.

CLI

AWS CLI

To create a CloudFront public key

The following `create-public-key` example creates a CloudFront public key by providing the parameters in a JSON file named `pub-key-config.json`. Before you can use this command, you must have a PEM-encoded public key. For more information, see Step 1: Create an RSA key pair in the *Amazon CloudFront Developer Guide*.

```
aws cloudfront create-public-key \
    --public-key-config file://pub-key-config.json
```

Contents of `pub-key-config.json`:

```
{
```

```
    "CallerReference": "cli-example",
    "Name": "ExampleKey",
    "EncodedKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxPMbCA2Ks0lnd7IR+3pw
\nwd3H/7jPGwj8bLUmore7bX+oeGpZ6QmLAe/1UOWcmZX2u70dYcSIzB1ofZtcn4cJ
\nenHBAzO3ohBY/L1tQGJfS2A+omnN6H16VZE1JCK8XSJyfze7MDLcUyHZETdxuvRb
\nA9X343/vMAuQPnhinFJ8Wdy8YBXSPpy7r95ylUQd9LfYTBzVZYG2tSesplcOkjM3\n2Uu
+oMWxQAw1NINnSLPinMVsutJy6ZqlV3McWNWe4T+STGtWhrPNqJEn45sIcCx4\nq
+kGZ2NQ0FyIyT2eiLKOX5Rgb/a36E/aMk4VoDsaenBQgG7WLTnstb9sr7MIhS6A\nrwIDAQAB\n-----
END PUBLIC KEY-----\n",
    "Comment": "example public key"
}
```

Output:

```
{
    "Location": "https://cloudfront.amazonaws.com/2019-03-26/public-key/
KDFB19YGCR002",
    "ETag": "E2QWRUHEXAMPLE",
    "PublicKey": {
        "Id": "KDFB19YGCR002",
        "CreatedTime": "2019-12-05T18:51:43.781Z",
        "PublicKeyConfig": {
            "CallerReference": "cli-example",
            "Name": "ExampleKey",
            "EncodedKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxPMbCA2Ks0lnd7IR+3pw
\nwd3H/7jPGwj8bLUmore7bX+oeGpZ6QmLAe/1UOWcmZX2u70dYcSIzB1ofZtcn4cJ
\nenHBAzO3ohBY/L1tQGJfS2A+omnN6H16VZE1JCK8XSJyfze7MDLcUyHZETdxuvRb
\nA9X343/vMAuQPnhinFJ8Wdy8YBXSPpy7r95ylUQd9LfYTBzVZYG2tSesplcOkjM3\n2Uu
+oMWxQAw1NINnSLPinMVsutJy6ZqlV3McWNWe4T+STGtWhrPNqJEn45sIcCx4\nq
+kGZ2NQ0FyIyT2eiLKOX5Rgb/a36E/aMk4VoDsaenBQgG7WLTnstb9sr7MIhS6A\nrwIDAQAB\n-----
END PUBLIC KEY-----\n",
            "Comment": "example public key"
        }
    }
}
```

For more information, see Step 2: Add your public key to CloudFront in the *Amazon CloudFront Developer Guide*.

- For API details, see CreatePublicKey in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The following code example reads in a public key and uploads it to Amazon CloudFront.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import software.amazon.awssdk.services.cloudfront.model.CreatePublicKeyResponse;
import software.amazon.awssdk.utils.IoUtils;

import java.io.IOException;
import java.io.InputStream;
import java.util.UUID;

public class CreatePublicKey {
    private static final Logger logger =
 LoggerFactory.getLogger(CreatePublicKey.class);

    public static String createPublicKey(CloudFrontClient cloudFrontClient,
 String publicKeyFileName) {
        try (InputStream is =
 CreatePublicKey.class.getClassLoader().getResourceAsStream(publicKeyFileName)) {
            String publicKeyString = IoUtils.toUtf8String(is);
            CreatePublicKeyResponse createPublicKeyResponse = cloudFrontClient
                    .createPublicKey(b -> b.publicKeyConfig(c -> c
                            .name("JavaCreatedPublicKey" + UUID.randomUUID())
                            .encodedKey(publicKeyString)
                            .callerReference(UUID.randomUUID().toString())));
            String createdPublicKeyId = createPublicKeyResponse.publicKey().id();
            logger.info("Public key created with id: [{}]", createdPublicKeyId);
            return createdPublicKeyId;

        } catch (IOException e) {
            throw new RuntimeException(e);
        }
```

```
    }
}
```

- For API details, see CreatePublicKey in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteDistribution` with an AWS SDK or CLI

The following code examples show how to use `DeleteDistribution`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get started with a basic content distribution network

CLI

AWS CLI

To delete a CloudFront distribution

The following `delete-distribution` example deletes the CloudFront distribution with the ID `EDFDVBD6EXAMPLE`. Before you can delete a distribution, you must disable it. To disable a distribution, use the update-distribution command. For more information, see the update-distribution examples.

When a distribution is disabled, you can delete it. To delete a distribution, you must use the `--if-match` option to provide the distribution's `ETag`. To get the `ETag`, use the get-distribution or get-distribution-config command.

```
aws cloudfront delete-distribution \
    --id EDFDVBD6EXAMPLE \
    --if-match E2QWRUHEXAMPLE
```

This command produces no output.

When successful, this command has no output.

For more information, see Delete a distribution in the *Amazon CloudFront Developer Guide*.

- For API details, see DeleteDistribution in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The following code example updates a distribution to *disabled*, uses a waiter that waits for the change to be deployed, then deletes the distribution.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.internal.waiters.ResponseOrException;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import
 software.amazon.awssdk.services.cloudfront.model.DeleteDistributionResponse;
import software.amazon.awssdk.services.cloudfront.model.DistributionConfig;
import software.amazon.awssdk.services.cloudfront.model.GetDistributionResponse;
import software.amazon.awssdk.services.cloudfront.waiters.CloudFrontWaiter;

public class DeleteDistribution {
        private static final Logger logger =
 LoggerFactory.getLogger(DeleteDistribution.class);

        public static void deleteDistribution(final CloudFrontClient
 cloudFrontClient, final String distributionId) {
                // First, disable the distribution by updating it.
                GetDistributionResponse response =
 cloudFrontClient.getDistribution(b -> b
                                .id(distributionId));
                String etag = response.eTag();
                DistributionConfig distConfig =
 response.distribution().distributionConfig();

                cloudFrontClient.updateDistribution(builder -> builder
```

```
                                .id(distributionId)
                                .distributionConfig(builder1 -> builder1

.cacheBehaviors(distConfig.cacheBehaviors())

.defaultCacheBehavior(distConfig.defaultCacheBehavior())
                                                    .enabled(false)
                                                    .origins(distConfig.origins())
                                                    .comment(distConfig.comment())

.callerReference(distConfig.callerReference())

.defaultCacheBehavior(distConfig.defaultCacheBehavior())

.priceClass(distConfig.priceClass())
                                                    .aliases(distConfig.aliases())
                                                    .logging(distConfig.logging())

.defaultRootObject(distConfig.defaultRootObject())

.customErrorResponses(distConfig.customErrorResponses())

.httpVersion(distConfig.httpVersion())

.isIPV6Enabled(distConfig.isIPV6Enabled())

.restrictions(distConfig.restrictions())

.viewerCertificate(distConfig.viewerCertificate())
                                                    .webACLId(distConfig.webACLId())

.originGroups(distConfig.originGroups()))
                                .ifMatch(etag));

            logger.info("Distribution [{}] is DISABLED, waiting for
deployment before deleting ...",
                                distributionId);
            GetDistributionResponse distributionResponse;
            try (CloudFrontWaiter cfWaiter =
CloudFrontWaiter.builder().client(cloudFrontClient).build()) {
                    ResponseOrException<GetDistributionResponse>
responseOrException = cfWaiter
                                        .waitUntilDistributionDeployed(builder ->
builder.id(distributionId)).matched();
```

```
                        distributionResponse = responseOrException.response()
                                                .orElseThrow(() -> new
 RuntimeException("Could not disable distribution"));
                }

                DeleteDistributionResponse deleteDistributionResponse =
 cloudFrontClient
                                .deleteDistribution(builder -> builder
                                                .id(distributionId)

 .ifMatch(distributionResponse.eTag()));
                if (deleteDistributionResponse.sdkHttpResponse().isSuccessful())
 {
                        logger.info("Distribution [{}] DELETED", distributionId);
                }
        }
}
```

- For API details, see DeleteDistribution in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetCloudFrontOriginAccessIdentity` with a CLI

The following code examples show how to use `GetCloudFrontOriginAccessIdentity`.

CLI

AWS CLI

To get a CloudFront origin access identity

The following `get-cloud-front-origin-access-identity` example gets the CloudFront origin access identity (OAI) with the ID `E74FTE3AEXAMPLE`, including its `ETag` and the associated S3 canonical ID. The OAI ID is returned in the output of the create-cloud-front-origin-access-identity and list-cloud-front-origin-access-identities commands.

```
aws cloudfront get-cloud-front-origin-access-identity \
```

```
    --id E74FTE3AEXAMPLE
```

Output:

```
{
    "ETag": "E2QWRUHEXAMPLE",
    "CloudFrontOriginAccessIdentity": {
        "Id": "E74FTE3AEXAMPLE",
        "S3CanonicalUserId":
 "cd13868f797c227fbea2830611a26fe0a21ba1b826ab4bed9b7771c9aEXAMPLE",
        "CloudFrontOriginAccessIdentityConfig": {
            "CallerReference": "cli-example",
            "Comment": "Example OAI"
        }
    }
}
```

For more information, see Restrict access to an Amazon S3 origin in the *Amazon CloudFront Developer Guide*.

- For API details, see GetCloudFrontOriginAccessIdentity in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a specific Amazon CloudFront origin access identity, specified by the -Id parameter. Although the -Id parameter is not required, if you do not specify it, no results are returned.

```
Get-CFCloudFrontOriginAccessIdentity -Id E3XXXXXXXXXXRT
```

Output:

```
     CloudFrontOriginAccessIdentityConfig    Id
    S3CanonicalUserId
     ------------------------------------    --
    -----------------
     Amazon.CloudFront.Model.CloudFrontOr... E3XXXXXXXXXXRT
    4b6e...
```

- For API details, see GetCloudFrontOriginAccessIdentity in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a specific Amazon CloudFront origin access identity, specified by the -Id parameter. Although the -Id parameter is not required, if you do not specify it, no results are returned.

```
Get-CFCloudFrontOriginAccessIdentity -Id E3XXXXXXXXXXRT
```

Output:

```
     CloudFrontOriginAccessIdentityConfig    Id
    S3CanonicalUserId
     ------------------------------------    --
    -----------------
     Amazon.CloudFront.Model.CloudFrontOr... E3XXXXXXXXXXRT
    4b6e...
```

- For API details, see GetCloudFrontOriginAccessIdentity in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetCloudFrontOriginAccessIdentityConfig` with a CLI

The following code examples show how to use `GetCloudFrontOriginAccessIdentityConfig`.

CLI

AWS CLI

To get a CloudFront origin access identity configuration

The following `get-cloud-front-origin-access-identity-config` example gets metadata about the CloudFront origin access identity (OAI) with the ID `E74FTE3AEXAMPLE`, including its `ETag`. The OAI ID is returned in the output of the create-cloud-front-origin-access-identity and list-cloud-front-origin-access-identities commands.

```
aws cloudfront get-cloud-front-origin-access-identity-config \
    --id E74FTE3AEXAMPLE
```

Output:

```
{
    "ETag": "E2QWRUHEXAMPLE",
    "CloudFrontOriginAccessIdentityConfig": {
        "CallerReference": "cli-example",
        "Comment": "Example OAI"
    }
}
```

For more information, see Restrict access to an Amazon S3 origin in the *Amazon CloudFront Developer Guide*.

- For API details, see GetCloudFrontOriginAccessIdentityConfig in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns configuration information about a single Amazon CloudFront origin access identity, specified by the -Id parameter. Errors occur if no -Id parameter is specified..

```
Get-CFCloudFrontOriginAccessIdentityConfig -Id E3XXXXXXXXXXRT
```

Output:

```
     CallerReference                                                Comment
     ---------------                                                -------
     mycallerreference: 2/1/2011 1:16:32 PM                         Caller
reference: 2/1/2011 1:16:32 PM
```

- For API details, see GetCloudFrontOriginAccessIdentityConfig in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns configuration information about a single Amazon CloudFront origin access identity, specified by the -Id parameter. Errors occur if no -Id parameter is specified..

```
Get-CFCloudFrontOriginAccessIdentityConfig -Id E3XXXXXXXXXXRT
```

Output:

```
     CallerReference                                          Comment
     ---------------                                          -------
     mycallerreference: 2/1/2011 1:16:32 PM                   Caller
reference: 2/1/2011 1:16:32 PM
```

- For API details, see GetCloudFrontOriginAccessIdentityConfig in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetDistribution` with a CLI

The following code examples show how to use `GetDistribution`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get started with a basic content distribution network

CLI

AWS CLI

To get a CloudFront distribution

The following `get-distribution` example gets the CloudFront distribution with the ID `EDFDVBD6EXAMPLE`, including its `ETag`. The distribution ID is returned in the create-distribution and list-distributions commands.

```
aws cloudfront get-distribution \
    --id EDFDVBD6EXAMPLE
```

Output:

```
{
    "ETag": "E2QWRUHEXAMPLE",
    "Distribution": {
        "Id": "EDFDVBD6EXAMPLE",
        "ARN": "arn:aws:cloudfront::123456789012:distribution/EDFDVBD6EXAMPLE",
        "Status": "Deployed",
        "LastModifiedTime": "2019-12-04T23:35:41.433Z",
        "InProgressInvalidationBatches": 0,
        "DomainName": "d111111abcdef8.cloudfront.net",
        "ActiveTrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "DistributionConfig": {
            "CallerReference": "cli-example",
            "Aliases": {
                "Quantity": 0
            },
            "DefaultRootObject": "index.html",
            "Origins": {
                "Quantity": 1,
                "Items": [
                    {
                        "Id": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-example",
                        "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                        "OriginPath": "",
                        "CustomHeaders": {
                            "Quantity": 0
                        },
                        "S3OriginConfig": {
                            "OriginAccessIdentity": ""
                        }
                    }
                ]
            },
            "OriginGroups": {
                "Quantity": 0
            },
```

```
            "DefaultCacheBehavior": {
                "TargetOriginId": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-
example",
                "ForwardedValues": {
                    "QueryString": false,
                    "Cookies": {
                        "Forward": "none"
                    },
                    "Headers": {
                        "Quantity": 0
                    },
                    "QueryStringCacheKeys": {
                        "Quantity": 0
                    }
                },
                "TrustedSigners": {
                    "Enabled": false,
                    "Quantity": 0
                },
                "ViewerProtocolPolicy": "allow-all",
                "MinTTL": 0,
                "AllowedMethods": {
                    "Quantity": 2,
                    "Items": [
                        "HEAD",
                        "GET"
                    ],
                    "CachedMethods": {
                        "Quantity": 2,
                        "Items": [
                            "HEAD",
                            "GET"
                        ]
                    }
                },
                "SmoothStreaming": false,
                "DefaultTTL": 86400,
                "MaxTTL": 31536000,
                "Compress": false,
                "LambdaFunctionAssociations": {
                    "Quantity": 0
                },
                "FieldLevelEncryptionId": ""
            },
```

```
            "CacheBehaviors": {
                "Quantity": 0
            },
            "CustomErrorResponses": {
                "Quantity": 0
            },
            "Comment": "",
            "Logging": {
                "Enabled": false,
                "IncludeCookies": false,
                "Bucket": "",
                "Prefix": ""
            },
            "PriceClass": "PriceClass_All",
            "Enabled": true,
            "ViewerCertificate": {
                "CloudFrontDefaultCertificate": true,
                "MinimumProtocolVersion": "TLSv1",
                "CertificateSource": "cloudfront"
            },
            "Restrictions": {
                "GeoRestriction": {
                    "RestrictionType": "none",
                    "Quantity": 0
                }
            },
            "WebACLId": "",
            "HttpVersion": "http2",
            "IsIPV6Enabled": true
        }
    }
}
```

For more information, see Configure distributions in the *Amazon CloudFront Developer Guide*.

- For API details, see GetDistribution in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Retrieves the information for a specific distribution.

```
Get-CFDistribution -Id EXAMPLE0000ID
```

- For API details, see GetDistribution in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Retrieves the information for a specific distribution.

```
Get-CFDistribution -Id EXAMPLE0000ID
```

- For API details, see GetDistribution in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetDistributionConfig` with an AWS SDK or CLI

The following code examples show how to use `GetDistributionConfig`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get started with a basic content distribution network

CLI

AWS CLI

To get a CloudFront distribution configuration

The following `get-distribution-config` example gets metadata about the CloudFront distribution with the ID `EDFDVBD6EXAMPLE`, including its `ETag`. The distribution ID is returned in the create-distribution and list-distributions commands.

```
aws cloudfront get-distribution-config \
    --id EDFDVBD6EXAMPLE
```

Output:

```
{
```

```
    "ETag": "E2QWRUHEXAMPLE",
    "DistributionConfig": {
        "CallerReference": "cli-example",
        "Aliases": {
            "Quantity": 0
        },
        "DefaultRootObject": "index.html",
        "Origins": {
            "Quantity": 1,
            "Items": [
                {
                    "Id": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-example",
                    "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                    "OriginPath": "",
                    "CustomHeaders": {
                        "Quantity": 0
                    },
                    "S3OriginConfig": {
                        "OriginAccessIdentity": ""
                    }
                }
            ]
        },
        "OriginGroups": {
            "Quantity": 0
        },
        "DefaultCacheBehavior": {
            "TargetOriginId": "amzn-s3-demo-bucket.s3.amazonaws.com-cli-example",
            "ForwardedValues": {
                "QueryString": false,
                "Cookies": {
                    "Forward": "none"
                },
                "Headers": {
                    "Quantity": 0
                },
                "QueryStringCacheKeys": {
                    "Quantity": 0
                }
            },
            "TrustedSigners": {
                "Enabled": false,
                "Quantity": 0
            },
```

```
            "ViewerProtocolPolicy": "allow-all",
            "MinTTL": 0,
            "AllowedMethods": {
                "Quantity": 2,
                "Items": [
                    "HEAD",
                    "GET"
                ],
                "CachedMethods": {
                    "Quantity": 2,
                    "Items": [
                        "HEAD",
                        "GET"
                    ]
                }
            },
            "SmoothStreaming": false,
            "DefaultTTL": 86400,
            "MaxTTL": 31536000,
            "Compress": false,
            "LambdaFunctionAssociations": {
                "Quantity": 0
            },
            "FieldLevelEncryptionId": ""
        },
        "CacheBehaviors": {
            "Quantity": 0
        },
        "CustomErrorResponses": {
            "Quantity": 0
        },
        "Comment": "",
        "Logging": {
            "Enabled": false,
            "IncludeCookies": false,
            "Bucket": "",
            "Prefix": ""
        },
        "PriceClass": "PriceClass_All",
        "Enabled": true,
        "ViewerCertificate": {
            "CloudFrontDefaultCertificate": true,
            "MinimumProtocolVersion": "TLSv1",
            "CertificateSource": "cloudfront"
```

```
        },
        "Restrictions": {
            "GeoRestriction": {
                "RestrictionType": "none",
                "Quantity": 0
            }
        },
        "WebACLId": "",
        "HttpVersion": "http2",
        "IsIPV6Enabled": true
    }
}
```

For more information, see Configure distributions in the *Amazon CloudFront Developer Guide*.

- For API details, see GetDistributionConfig in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Retrieves the configuration for a specific distribution.

```
Get-CFDistributionConfig -Id EXAMPLE0000ID
```

- For API details, see GetDistributionConfig in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Retrieves the configuration for a specific distribution.

```
Get-CFDistributionConfig -Id EXAMPLE0000ID
```

- For API details, see GetDistributionConfig in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudFrontWrapper:
    """Encapsulates Amazon CloudFront operations."""

    def __init__(self, cloudfront_client):
        """
        :param cloudfront_client: A Boto3 CloudFront client
        """
        self.cloudfront_client = cloudfront_client


    def update_distribution(self):
        distribution_id = input(
            "This script updates the comment for a CloudFront distribution.\n"
            "Enter a CloudFront distribution ID: "
        )

        distribution_config_response =
 self.cloudfront_client.get_distribution_config(
            Id=distribution_id
        )
        distribution_config = distribution_config_response["DistributionConfig"]
        distribution_etag = distribution_config_response["ETag"]

        distribution_config["Comment"] = input(
            f"\nThe current comment for distribution {distribution_id} is "
            f"'{distribution_config['Comment']}'.\n"
            f"Enter a new comment: "
        )
        self.cloudfront_client.update_distribution(
            DistributionConfig=distribution_config,
            Id=distribution_id,
            IfMatch=distribution_etag,
```

```
        )
        print("Done!")
```

- For API details, see GetDistributionConfig in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListCloudFrontOriginAccessIdentities` with a CLI

The following code examples show how to use `ListCloudFrontOriginAccessIdentities`.

CLI

AWS CLI

To list CloudFront origin access identities

The following `list-cloud-front-origin-access-identities` example gets a list of the CloudFront origin access identities (OAIs) in your AWS account.

```
aws cloudfront list-cloud-front-origin-access-identities
```

Output:

```
{
    "CloudFrontOriginAccessIdentityList": {
        "Items": [
            {
                "Id": "E74FTE3AEXAMPLE",
                "S3CanonicalUserId":
 "cd13868f797c227fbea2830611a26fe0a21ba1b826ab4bed9b7771c9aEXAMPLE",
                "Comment": "Example OAI"
            },
            {
                "Id": "EH1HDMBEXAMPLE",
                "S3CanonicalUserId":
 "1489f6f2e6faacaae7ff64c4c3e6956c24f78788abfc1718c3527c263bf7a17EXAMPLE",
```

```
                "Comment": "Test OAI"
            },
            {
                "Id": "E2X2C9TEXAMPLE",
                "S3CanonicalUserId":
 "cbfeebb915a64749f9be546a45b3fcfd3a31c779673c13c4dd460911ae402c2EXAMPLE",
                "Comment": "Example OAI #2"
            }
        ]
    }
}
```

For more information, see Restrict access to an Amazon S3 origin in the *Amazon CloudFront Developer Guide*.

- For API details, see ListCloudFrontOriginAccessIdentities in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: This example returns a list of Amazon CloudFront origin access identities. Because the -MaxItem parameter specifies a value of 2, the results include two identities.

```
Get-CFCloudFrontOriginAccessIdentityList -MaxItem 2
```

Output:

```
IsTruncated : True
Items       : {E326XXXXXXXXXT, E1YWXXXXXXX9B}
Marker      :
MaxItems    : 2
NextMarker  : E1YXXXXXXXXX9B
Quantity    : 2
```

- For API details, see ListCloudFrontOriginAccessIdentities in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: This example returns a list of Amazon CloudFront origin access identities. Because the -MaxItem parameter specifies a value of 2, the results include two identities.

```
Get-CFCloudFrontOriginAccessIdentityList -MaxItem 2
```

Output:

```
IsTruncated : True
Items       : {E326XXXXXXXXXT, E1YWXXXXXXX9B}
Marker      :
MaxItems    : 2
NextMarker  : E1YXXXXXXXXX9B
Quantity    : 2
```

- For API details, see ListCloudFrontOriginAccessIdentities in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListDistributions` with an AWS SDK or CLI

The following code examples show how to use `ListDistributions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Getting started with web application firewalls

CLI

AWS CLI

To list CloudFront distributions

The following `list-distributions` example gets a list of the CloudFront distributions in your AWS account.

```
aws cloudfront list-distributions
```

Output:

```
{
    "DistributionList": {
        "Items": [
            {
                "Id": "E23YS8OEXAMPLE",
                "ARN": "arn:aws:cloudfront::123456789012:distribution/
E23YS8OEXAMPLE",
                "Status": "Deployed",
                "LastModifiedTime": "2024-08-05T18:23:40.375000+00:00",
                "DomainName": "abcdefgh12ijk.cloudfront.net",
                "Aliases": {
                    "Quantity": 0
                },
                "Origins": {
                    "Quantity": 1,
                    "Items": [
                        {
                            "Id": "amzn-s3-demo-bucket.s3.us-
west-2.amazonaws.com",
                            "DomainName": "amzn-s3-demo-bucket.s3.us-
west-2.amazonaws.com",
                            "OriginPath": "",
                            "CustomHeaders": {
                                "Quantity": 0
                            },
                            "S3OriginConfig": {
                                "OriginAccessIdentity": ""
                            },
                            "ConnectionAttempts": 3,
                            "ConnectionTimeout": 10,
                            "OriginShield": {
                                "Enabled": false
                            },
                            "OriginAccessControlId": "EIAP8PEXAMPLE"
                        }
                    ]
                },
                "OriginGroups": {
                    "Quantity": 0
                },
                "DefaultCacheBehavior": {
                    "TargetOriginId": "amzn-s3-demo-bucket.s3.us-
west-2.amazonaws.com",
```

```
                    "TrustedSigners": {
                        "Enabled": false,
                        "Quantity": 0
                    },
                    "TrustedKeyGroups": {
                        "Enabled": false,
                        "Quantity": 0
                    },
                    "ViewerProtocolPolicy": "allow-all",
                    "AllowedMethods": {
                        "Quantity": 2,
                        "Items": [
                            "HEAD",
                            "GET"
                        ],
                        "CachedMethods": {
                            "Quantity": 2,
                            "Items": [
                                "HEAD",
                                "GET"
                            ]
                        }
                    },
                    "SmoothStreaming": false,
                    "Compress": true,
                    "LambdaFunctionAssociations": {
                        "Quantity": 0
                    },
                    "FunctionAssociations": {
                        "Quantity": 0
                    },
                    "FieldLevelEncryptionId": "",
                    "CachePolicyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                },
                "CacheBehaviors": {
                    "Quantity": 0
                },
                "CustomErrorResponses": {
                    "Quantity": 0
                },
                "Comment": "",
                "PriceClass": "PriceClass_All",
                "Enabled": true,
                "ViewerCertificate": {
```

```
                    "CloudFrontDefaultCertificate": true,
                    "SSLSupportMethod": "vip",
                    "MinimumProtocolVersion": "TLSv1",
                    "CertificateSource": "cloudfront"
                },
                "Restrictions": {
                    "GeoRestriction": {
                        "RestrictionType": "none",
                        "Quantity": 0
                    }
                },
                "WebACLId": "",
                "HttpVersion": "HTTP2",
                "IsIPV6Enabled": true,
                "Staging": false
            }
        ]
    }
}
```

For more information, see Configure distributions in the *Amazon CloudFront Developer Guide*.

- For API details, see ListDistributions in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Returns distributions.

```
Get-CFDistributionList
```

- For API details, see ListDistributions in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Returns distributions.

```
Get-CFDistributionList
```

- For API details, see ListDistributions in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudFrontWrapper:
    """Encapsulates Amazon CloudFront operations."""

    def __init__(self, cloudfront_client):
        """
        :param cloudfront_client: A Boto3 CloudFront client
        """
        self.cloudfront_client = cloudfront_client


    def list_distributions(self):
        print("CloudFront distributions:\n")
        distributions = self.cloudfront_client.list_distributions()
        if distributions["DistributionList"]["Quantity"] > 0:
            for distribution in distributions["DistributionList"]["Items"]:
                print(f"Domain: {distribution['DomainName']}")
                print(f"Distribution Id: {distribution['Id']}")
                print(
                    f"Certificate Source: "
                    f"{distribution['ViewerCertificate']['CertificateSource']}"
                )
                if distribution["ViewerCertificate"]["CertificateSource"] ==
 "acm":
                    print(
                        f"Certificate: {distribution['ViewerCertificate']
['Certificate']}"
                    )
                print("")
        else:
            print("No CloudFront distributions detected.")
```

- For API details, see ListDistributions in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        oo_result = lo_fnt->listdistributions( ). " oo_result is returned for
testing purposes. "
        MESSAGE 'Retrieved list of CloudFront distributions.' TYPE 'I'.
      CATCH /aws1/cx_fntinvalidargument.
        MESSAGE 'Invalid argument provided.' TYPE 'E'.
    ENDTRY.
```

- For API details, see ListDistributions in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateDistribution` with an AWS SDK or CLI

The following code examples show how to use `UpdateDistribution`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Get started with a basic content distribution network

CLI

AWS CLI

Example 1: To update a CloudFront distribution's default root object

The following `update-distribution` example updates the default root object to `index.html` for the CloudFront distribution with the ID `EDFDVBD6EXAMPLE`.

```
aws cloudfront update-distribution \
    --id EDFDVBD6EXAMPLE \
    --default-root-object index.html
```

Output:

```
{
    "ETag": "E2QWRUHEXAMPLE",
    "Distribution": {
        "Id": "EDFDVBD6EXAMPLE",
        "ARN": "arn:aws:cloudfront::123456789012:distribution/EDFDVBD6EXAMPLE",
        "Status": "InProgress",
        "LastModifiedTime": "2019-12-06T18:55:39.870Z",
        "InProgressInvalidationBatches": 0,
        "DomainName": "d111111abcdef8.cloudfront.net",
        "ActiveTrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "DistributionConfig": {
            "CallerReference": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "Aliases": {
                "Quantity": 0
            },
            "DefaultRootObject": "index.html",
            "Origins": {
                "Quantity": 1,
                "Items": [
                    {
                        "Id": "example-website",
                        "DomainName": "www.example.com",
                        "OriginPath": "",
                        "CustomHeaders": {
                            "Quantity": 0
```

```
                    },
                    "CustomOriginConfig": {
                        "HTTPPort": 80,
                        "HTTPSPort": 443,
                        "OriginProtocolPolicy": "match-viewer",
                        "OriginSslProtocols": {
                            "Quantity": 2,
                            "Items": [
                                "SSLv3",
                                "TLSv1"
                            ]
                        },
                        "OriginReadTimeout": 30,
                        "OriginKeepaliveTimeout": 5
                    }
                }
            ]
        },
        "OriginGroups": {
            "Quantity": 0
        },
        "DefaultCacheBehavior": {
            "TargetOriginId": "example-website",
            "ForwardedValues": {
                "QueryString": false,
                "Cookies": {
                    "Forward": "none"
                },
                "Headers": {
                    "Quantity": 1,
                    "Items": [
                        "*"
                    ]
                },
                "QueryStringCacheKeys": {
                    "Quantity": 0
                }
            },
            "TrustedSigners": {
                "Enabled": false,
                "Quantity": 0
            },
            "ViewerProtocolPolicy": "allow-all",
            "MinTTL": 0,
```

```
                "AllowedMethods": {
                    "Quantity": 2,
                    "Items": [
                        "HEAD",
                        "GET"
                    ],
                    "CachedMethods": {
                        "Quantity": 2,
                        "Items": [
                            "HEAD",
                            "GET"
                        ]
                    }
                },
                "SmoothStreaming": false,
                "DefaultTTL": 86400,
                "MaxTTL": 31536000,
                "Compress": false,
                "LambdaFunctionAssociations": {
                    "Quantity": 0
                },
                "FieldLevelEncryptionId": ""
            },
            "CacheBehaviors": {
                "Quantity": 0
            },
            "CustomErrorResponses": {
                "Quantity": 0
            },
            "Comment": "",
            "Logging": {
                "Enabled": false,
                "IncludeCookies": false,
                "Bucket": "",
                "Prefix": ""
            },
            "PriceClass": "PriceClass_All",
            "Enabled": true,
            "ViewerCertificate": {
                "CloudFrontDefaultCertificate": true,
                "MinimumProtocolVersion": "TLSv1",
                "CertificateSource": "cloudfront"
            },
            "Restrictions": {
```

```
                "GeoRestriction": {
                    "RestrictionType": "none",
                    "Quantity": 0
                }
            },
            "WebACLId": "",
            "HttpVersion": "http1.1",
            "IsIPV6Enabled": true
        }
    }
}
```

For more information, see Update a distribution in the *Amazon CloudFront Developer Guide*.

Example 2: To update a CloudFront distribution

The following `update-distribution` example disables the CloudFront distribution with the ID `EMLARXS9EXAMPLE` by providing the distribution configuration in a JSON file named `dist-config-disable.json`. To update a distribution, you must use the `--if-match` option to provide the distribution's `ETag`. To get the `ETag`, use the get-distribution or get-distribution-config command. Note that the `Enabled` field is set to `false` in the JSON file.

After you use the following example to disable a distribution, you can use the delete-distribution command to delete it.

```
aws cloudfront update-distribution \
    --id EMLARXS9EXAMPLE \
    --if-match E2QWRUHEXAMPLE \
    --distribution-config file://dist-config-disable.json
```

Contents of `dist-config-disable.json`:

```
{
    "CallerReference": "cli-1574382155-496510",
    "Aliases": {
        "Quantity": 0
    },
    "DefaultRootObject": "index.html",
    "Origins": {
        "Quantity": 1,
        "Items": [
            {
```

```
                "Id": "amzn-s3-demo-bucket.s3.amazonaws.com-1574382155-273939",
                "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                "OriginPath": "",
                "CustomHeaders": {
                    "Quantity": 0
                },
                "S3OriginConfig": {
                    "OriginAccessIdentity": ""
                }
            }
        ]
    },
    "OriginGroups": {
        "Quantity": 0
    },
    "DefaultCacheBehavior": {
        "TargetOriginId": "amzn-s3-demo-
bucket.s3.amazonaws.com-1574382155-273939",
        "ForwardedValues": {
            "QueryString": false,
            "Cookies": {
                "Forward": "none"
            },
            "Headers": {
                "Quantity": 0
            },
            "QueryStringCacheKeys": {
                "Quantity": 0
            }
        },
        "TrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "ViewerProtocolPolicy": "allow-all",
        "MinTTL": 0,
        "AllowedMethods": {
            "Quantity": 2,
            "Items": [
                "HEAD",
                "GET"
            ],
            "CachedMethods": {
                "Quantity": 2,
```

```
                "Items": [
                    "HEAD",
                    "GET"
                ]
            }
        },
        "SmoothStreaming": false,
        "DefaultTTL": 86400,
        "MaxTTL": 31536000,
        "Compress": false,
        "LambdaFunctionAssociations": {
            "Quantity": 0
        },
        "FieldLevelEncryptionId": ""
    },
    "CacheBehaviors": {
        "Quantity": 0
    },
    "CustomErrorResponses": {
        "Quantity": 0
    },
    "Comment": "",
    "Logging": {
        "Enabled": false,
        "IncludeCookies": false,
        "Bucket": "",
        "Prefix": ""
    },
    "PriceClass": "PriceClass_All",
    "Enabled": false,
    "ViewerCertificate": {
        "CloudFrontDefaultCertificate": true,
        "MinimumProtocolVersion": "TLSv1",
        "CertificateSource": "cloudfront"
    },
    "Restrictions": {
        "GeoRestriction": {
            "RestrictionType": "none",
            "Quantity": 0
        }
    },
    "WebACLId": "",
    "HttpVersion": "http2",
    "IsIPV6Enabled": true
```

```
}
```

Output:

```
{
    "ETag": "E9LHASXEXAMPLE",
    "Distribution": {
        "Id": "EMLARXS9EXAMPLE",
        "ARN": "arn:aws:cloudfront::123456789012:distribution/EMLARXS9EXAMPLE",
        "Status": "InProgress",
        "LastModifiedTime": "2019-12-06T18:32:35.553Z",
        "InProgressInvalidationBatches": 0,
        "DomainName": "d111111abcdef8.cloudfront.net",
        "ActiveTrustedSigners": {
            "Enabled": false,
            "Quantity": 0
        },
        "DistributionConfig": {
            "CallerReference": "cli-1574382155-496510",
            "Aliases": {
                "Quantity": 0
            },
            "DefaultRootObject": "index.html",
            "Origins": {
                "Quantity": 1,
                "Items": [
                    {
                        "Id": "amzn-s3-demo-
bucket.s3.amazonaws.com-1574382155-273939",
                        "DomainName": "amzn-s3-demo-bucket.s3.amazonaws.com",
                        "OriginPath": "",
                        "CustomHeaders": {
                            "Quantity": 0
                        },
                        "S3OriginConfig": {
                            "OriginAccessIdentity": ""
                        }
                    }
                ]
            },
            "OriginGroups": {
                "Quantity": 0
            },
```

```
            "DefaultCacheBehavior": {
                "TargetOriginId": "amzn-s3-demo-
bucket.s3.amazonaws.com-1574382155-273939",
                "ForwardedValues": {
                    "QueryString": false,
                    "Cookies": {
                        "Forward": "none"
                    },
                    "Headers": {
                        "Quantity": 0
                    },
                    "QueryStringCacheKeys": {
                        "Quantity": 0
                    }
                },
                "TrustedSigners": {
                    "Enabled": false,
                    "Quantity": 0
                },
                "ViewerProtocolPolicy": "allow-all",
                "MinTTL": 0,
                "AllowedMethods": {
                    "Quantity": 2,
                    "Items": [
                        "HEAD",
                        "GET"
                    ],
                    "CachedMethods": {
                        "Quantity": 2,
                        "Items": [
                            "HEAD",
                            "GET"
                        ]
                    }
                },
                "SmoothStreaming": false,
                "DefaultTTL": 86400,
                "MaxTTL": 31536000,
                "Compress": false,
                "LambdaFunctionAssociations": {
                    "Quantity": 0
                },
                "FieldLevelEncryptionId": ""
            },
```

```
            "CacheBehaviors": {
                "Quantity": 0
            },
            "CustomErrorResponses": {
                "Quantity": 0
            },
            "Comment": "",
            "Logging": {
                "Enabled": false,
                "IncludeCookies": false,
                "Bucket": "",
                "Prefix": ""
            },
            "PriceClass": "PriceClass_All",
            "Enabled": false,
            "ViewerCertificate": {
                "CloudFrontDefaultCertificate": true,
                "MinimumProtocolVersion": "TLSv1",
                "CertificateSource": "cloudfront"
            },
            "Restrictions": {
                "GeoRestriction": {
                    "RestrictionType": "none",
                    "Quantity": 0
                }
            },
            "WebACLId": "",
            "HttpVersion": "http2",
            "IsIPV6Enabled": true
        }
    }
}
```

For more information, see Update a distribution in the *Amazon CloudFront Developer Guide*.

- For API details, see UpdateDistribution in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import software.amazon.awssdk.services.cloudfront.model.GetDistributionRequest;
import software.amazon.awssdk.services.cloudfront.model.GetDistributionResponse;
import software.amazon.awssdk.services.cloudfront.model.Distribution;
import software.amazon.awssdk.services.cloudfront.model.DistributionConfig;
import
 software.amazon.awssdk.services.cloudfront.model.UpdateDistributionRequest;
import software.amazon.awssdk.services.cloudfront.model.CloudFrontException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ModifyDistribution {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <id>\s

                Where:
                    id - the id value of the distribution.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String id = args[0];
        CloudFrontClient cloudFrontClient = CloudFrontClient.builder()
                .region(Region.AWS_GLOBAL)
                .build();

        modDistribution(cloudFrontClient, id);
        cloudFrontClient.close();
    }

    public static void modDistribution(CloudFrontClient cloudFrontClient, String
idVal) {
        try {
            // Get the Distribution to modify.
            GetDistributionRequest disRequest = GetDistributionRequest.builder()
                    .id(idVal)
                    .build();

            GetDistributionResponse response =
cloudFrontClient.getDistribution(disRequest);
            Distribution disObject = response.distribution();
            DistributionConfig config = disObject.distributionConfig();

            // Create a new DistributionConfig object and add new values to
comment and
            // aliases
            DistributionConfig config1 = DistributionConfig.builder()
                    .aliases(config.aliases()) // You can pass in new values here
                    .comment("New Comment")
                    .cacheBehaviors(config.cacheBehaviors())
                    .priceClass(config.priceClass())
                    .defaultCacheBehavior(config.defaultCacheBehavior())
                    .enabled(config.enabled())
                    .callerReference(config.callerReference())
                    .logging(config.logging())
                    .originGroups(config.originGroups())
                    .origins(config.origins())
                    .restrictions(config.restrictions())
                    .defaultRootObject(config.defaultRootObject())
                    .webACLId(config.webACLId())
                    .httpVersion(config.httpVersion())
                    .viewerCertificate(config.viewerCertificate())
```

```
                    .customErrorResponses(config.customErrorResponses())
                    .build();

            UpdateDistributionRequest updateDistributionRequest =
 UpdateDistributionRequest.builder()
                    .distributionConfig(config1)
                    .id(disObject.id())
                    .ifMatch(response.eTag())
                    .build();

            cloudFrontClient.updateDistribution(updateDistributionRequest);

        } catch (CloudFrontException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see UpdateDistribution in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CloudFrontWrapper:
    """Encapsulates Amazon CloudFront operations."""

    def __init__(self, cloudfront_client):
        """
        :param cloudfront_client: A Boto3 CloudFront client
        """
        self.cloudfront_client = cloudfront_client
```

```
    def update_distribution(self):
        distribution_id = input(
            "This script updates the comment for a CloudFront distribution.\n"
            "Enter a CloudFront distribution ID: "
        )

        distribution_config_response =
self.cloudfront_client.get_distribution_config(
            Id=distribution_id
        )
        distribution_config = distribution_config_response["DistributionConfig"]
        distribution_etag = distribution_config_response["ETag"]

        distribution_config["Comment"] = input(
            f"\nThe current comment for distribution {distribution_id} is "
            f"'{distribution_config['Comment']}'.\n"
            f"Enter a new comment: "
        )
        self.cloudfront_client.update_distribution(
            DistributionConfig=distribution_config,
            Id=distribution_id,
            IfMatch=distribution_etag,
        )
        print("Done!")
```

- For API details, see UpdateDistribution in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        " Get the current distribution configuration and ETag "
```

```
        DATA(lo_distribution_config_result) = lo_fnt-
>getdistributionconfig( iv_id = iv_distribution_id ).
        DATA(lo_old_config) = lo_distribution_config_result-
>get_distributionconfig( ).
        DATA(lv_etag) = lo_distribution_config_result->get_etag( ).

        " Create a new distribution config with the updated comment "
        " Since the config object is immutable, we need to create a new one with
 all existing values "
        DATA(lo_new_config) = NEW /aws1/cl_fntdistributionconfig(
          iv_callerreference = lo_old_config->get_callerreference( )
          io_aliases = lo_old_config->get_aliases( )
          iv_defaultrootobject = lo_old_config->get_defaultrootobject( )
          io_origins = lo_old_config->get_origins( )
          io_origingroups = lo_old_config->get_origingroups( )
          io_defaultcachebehavior = lo_old_config->get_defaultcachebehavior( )
          io_cachebehaviors = lo_old_config->get_cachebehaviors( )
          io_customerrorresponses = lo_old_config->get_customerrorresponses( )
          iv_comment = iv_comment
          io_logging = lo_old_config->get_logging( )
          iv_priceclass = lo_old_config->get_priceclass( )
          iv_enabled = lo_old_config->get_enabled( )
          io_viewercertificate = lo_old_config->get_viewercertificate( )
          io_restrictions = lo_old_config->get_restrictions( )
          iv_webaclid = lo_old_config->get_webaclid( )
          iv_httpversion = lo_old_config->get_httpversion( )
          iv_isipv6enabled = lo_old_config->get_isipv6enabled( ) ).

        " Update the distribution with the modified configuration "
        lo_fnt->updatedistribution(
          io_distributionconfig = lo_new_config
          iv_id = iv_distribution_id
          iv_ifmatch = lv_etag ).
        MESSAGE 'CloudFront distribution updated successfully.' TYPE 'I'.
      CATCH /aws1/cx_fntnosuchdistribution.
        MESSAGE 'Distribution does not exist.' TYPE 'E'.
      CATCH /aws1/cx_fntpreconditionfailed.
        MESSAGE 'Precondition failed - ETag mismatch.' TYPE 'E'.
      CATCH /aws1/cx_fntinvalidifmatchvrs.
        MESSAGE 'Invalid If-Match version.' TYPE 'E'.
    ENDTRY.
```

- For API details, see UpdateDistribution in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for CloudFront using AWS SDKs

The following code examples show you how to implement common scenarios in CloudFront with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within CloudFront or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Create SaaS manager resources AWS SDK
- Delete CloudFront signing resources using AWS SDK
- Get started with a basic content distribution network
- Getting started with web application firewalls
- Create signed URLs and cookies using an AWS SDK

Create SaaS manager resources AWS SDK

The following code example shows how to create a multi-tenant distribution and distribution tenant with various configurations.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The following example demonstrates how to create a multi-tenant distribution with parameters and wildcard certificate.

```
import software.amazon.awssdk.core.internal.waiters.ResponseOrException;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import software.amazon.awssdk.services.cloudfront.model.ConnectionMode;
import
 software.amazon.awssdk.services.cloudfront.model.CreateDistributionResponse;
import software.amazon.awssdk.services.cloudfront.model.Distribution;
import software.amazon.awssdk.services.cloudfront.model.GetDistributionResponse;
import software.amazon.awssdk.services.cloudfront.model.HttpVersion;
import software.amazon.awssdk.services.cloudfront.model.Method;
import software.amazon.awssdk.services.cloudfront.model.SSLSupportMethod;
import software.amazon.awssdk.services.cloudfront.model.ViewerProtocolPolicy;
import software.amazon.awssdk.services.cloudfront.waiters.CloudFrontWaiter;
import software.amazon.awssdk.services.s3.S3Client;

import java.time.Instant;

public class CreateMultiTenantDistribution {
    public static Distribution
 CreateMultiTenantDistributionWithCert(CloudFrontClient cloudFrontClient,
                                                                    S3Client
 s3Client,
                                                                    final String
 bucketName,
                                                                    final String
 certificateArn) {
        // fetch the origin info if necessary
        final String region = s3Client.headBucket(b ->
 b.bucket(bucketName)).sdkHttpResponse().headers()
                .get("x-amz-bucket-region").get(0);
        final String originDomain = bucketName + ".s3." + region +
 ".amazonaws.com";
        String originId = originDomain; // Use the originDomain value for the
 originId.

        CreateDistributionResponse createDistResponse =
 cloudFrontClient.createDistribution(builder -> builder
                .distributionConfig(b1 -> b1
                        .httpVersion(HttpVersion.HTTP2)
                        .enabled(true)
                        .comment("Template Distribution with cert built with
 java")
                        .connectionMode(ConnectionMode.TENANT_ONLY)
```

```
                        .callerReference(Instant.now().toString())
                        .viewerCertificate(certBuilder -> certBuilder
                                .acmCertificateArn(certificateArn)
                                .sslSupportMethod(SSLSupportMethod.SNI_ONLY))
                        .origins(b2 -> b2
                                .quantity(1)
                                .items(b3 -> b3
                                        .domainName(originDomain)
                                        .id(originId)
                                        .originPath("/{{tenantName}}")
                                        .s3OriginConfig(builder4 -> builder4
                                                .originAccessIdentity(
                                                        ""))))
                        .tenantConfig(b5 -> b5
                                .parameterDefinitions(b6 -> b6
                                        .name("tenantName")
                                        .definition(b7 -> b7
                                                .stringSchema(b8 -> b8
                                                        .comment("tenantName
 value")
                                                        .defaultValue("root")
                                                        .required(false)))))
                        .defaultCacheBehavior(b2 -> b2

 .viewerProtocolPolicy(ViewerProtocolPolicy.ALLOW_ALL)
                                .targetOriginId(originId)
                                .cachePolicyId("658327ea-f89d-4fab-
a63d-7e88639e58f6") // CachingOptimized Policy
                                .allowedMethods(b4 -> b4
                                        .quantity(2)
                                        .items(Method.HEAD, Method.GET)))
                ));

        final Distribution distribution = createDistResponse.distribution();
        try (CloudFrontWaiter cfWaiter =
 CloudFrontWaiter.builder().client(cloudFrontClient).build()) {
            ResponseOrException<GetDistributionResponse> responseOrException =
 cfWaiter
                    .waitUntilDistributionDeployed(builder ->
 builder.id(distribution.id()))
                    .matched();
            responseOrException.response()
                    .orElseThrow(() -> new RuntimeException("Distribution not
 created"));
```

```
        }
        return distribution;
    }

    public static Distribution
CreateMultiTenantDistributionNoCert(CloudFrontClient cloudFrontClient,
                                                            S3Client s3Client,
                                                            final String
bucketName) {
        // fetch the origin info if necessary
        final String region = s3Client.headBucket(b ->
b.bucket(bucketName)).sdkHttpResponse().headers()
                .get("x-amz-bucket-region").get(0);
        final String originDomain = bucketName + ".s3." + region +
".amazonaws.com";
        String originId = originDomain; // Use the originDomain value for the
originId.

        CreateDistributionResponse createDistResponse =
cloudFrontClient.createDistribution(builder -> builder
                .distributionConfig(b1 -> b1
                        .httpVersion(HttpVersion.HTTP2)
                        .enabled(true)
                        .comment("Template Distribution with cert built with
java")
                        .connectionMode(ConnectionMode.TENANT_ONLY)
                        .callerReference(Instant.now().toString())
                        .origins(b2 -> b2
                                .quantity(1)
                                .items(b3 -> b3
                                        .domainName(originDomain)
                                        .id(originId)
                                        .originPath("/{{tenantName}}")
                                        .s3OriginConfig(builder4 -> builder4
                                                .originAccessIdentity(
                                                        ""))))
                        .tenantConfig(b5 -> b5
                                .parameterDefinitions(b6 -> b6
                                        .name("tenantName")
                                        .definition(b7 -> b7
                                                .stringSchema(b8 -> b8
                                                        .comment("tenantName
value")
                                                        .defaultValue("root")
```

```
                                                                .required(false)))))
                        .defaultCacheBehavior(b2 -> b2

 .viewerProtocolPolicy(ViewerProtocolPolicy.ALLOW_ALL)
                                .targetOriginId(originId)
                                .cachePolicyId("658327ea-f89d-4fab-
a63d-7e88639e58f6") // CachingOptimized Policy
                                .allowedMethods(b4 -> b4
                                        .quantity(2)
                                        .items(Method.HEAD, Method.GET)))
                ));

        final Distribution distribution = createDistResponse.distribution();
        try (CloudFrontWaiter cfWaiter =
 CloudFrontWaiter.builder().client(cloudFrontClient).build()) {
            ResponseOrException<GetDistributionResponse> responseOrException =
 cfWaiter
                    .waitUntilDistributionDeployed(builder ->
 builder.id(distribution.id()))
                    .matched();
            responseOrException.response()
                    .orElseThrow(() -> new RuntimeException("Distribution not
 created"));
        }
        return distribution;
    }
}
```

The following example demonstrates how to create a distribution tenant associated with that template, including utilizing the parameter we declared above. Note that we don't need to add certificate info here because our domain is already covered by the parent template.

```
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import
 software.amazon.awssdk.services.cloudfront.model.CreateConnectionGroupResponse;
import
 software.amazon.awssdk.services.cloudfront.model.CreateDistributionTenantResponse;
import software.amazon.awssdk.services.cloudfront.model.DistributionTenant;
import
 software.amazon.awssdk.services.cloudfront.model.GetConnectionGroupResponse;
import software.amazon.awssdk.services.cloudfront.model.ValidationTokenHost;
```

```
import software.amazon.awssdk.services.route53.Route53Client;
import software.amazon.awssdk.services.route53.model.RRType;

import java.time.Instant;

public class CreateDistributionTenant {

    public static DistributionTenant
 createDistributionTenantNoCert(CloudFrontClient cloudFrontClient,
                                                                    Route53Client
 route53Client,
                                                                    String
 distributionId,
                                                                    String
 domain,
                                                                    String
 hostedZoneId) {
        CreateDistributionTenantResponse createResponse =
 cloudFrontClient.createDistributionTenant(builder -> builder
                .distributionId(distributionId)
                .domains(b1 -> b1
                        .domain(domain))
                .parameters(b2 -> b2
                        .name("tenantName")
                        .value("myTenant"))
                .enabled(false)
                .name("no-cert-tenant")
        );

        final DistributionTenant distributionTenant =
 createResponse.distributionTenant();

        // Then update the Route53 hosted zone to point your domain at the
 distribution tenant
        // We fetch the RoutingEndpoint to point to via the default connection
 group that was created for your tenant
        final GetConnectionGroupResponse fetchedConnectionGroup =
 cloudFrontClient.getConnectionGroup(builder -> builder
                .identifier(distributionTenant.connectionGroupId()));

        route53Client.changeResourceRecordSets(builder -> builder
                .hostedZoneId(hostedZoneId)
                .changeBatch(b1 -> b1
                        .comment("ChangeBatch comment")
```

```
                        .changes(b2 -> b2
                                .resourceRecordSet(b3 -> b3
                                        .name(domain)
                                        .type("CNAME")
                                        .ttl(300L)
                                        .resourceRecords(b4 -> b4

 .value(fetchedConnectionGroup.connectionGroup().routingEndpoint())))
                                .action("CREATE"))
                ));
        return distributionTenant;
    }

}
```

If the viewer certificate was omitted from the parent template, you would need to add certificate info on the tenant(s) associated with it instead. The following example demonstrates how to do so via an ACM certificate arn that covers the necessary domain for the tenant.

```
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import
 software.amazon.awssdk.services.cloudfront.model.CreateConnectionGroupResponse;
import
 software.amazon.awssdk.services.cloudfront.model.CreateDistributionTenantResponse;
import software.amazon.awssdk.services.cloudfront.model.DistributionTenant;
import
 software.amazon.awssdk.services.cloudfront.model.GetConnectionGroupResponse;
import software.amazon.awssdk.services.cloudfront.model.ValidationTokenHost;
import software.amazon.awssdk.services.route53.Route53Client;
import software.amazon.awssdk.services.route53.model.RRType;

import java.time.Instant;

public class CreateDistributionTenant {

    public static DistributionTenant
 createDistributionTenantWithCert(CloudFrontClient cloudFrontClient,

 Route53Client route53Client,
```

```
                                                                        String
distributionId,
                                                                        String
domain,
                                                                        String
hostedZoneId,
                                                                        String
certificateArn) {
        CreateDistributionTenantResponse createResponse =
cloudFrontClient.createDistributionTenant(builder -> builder
                .distributionId(distributionId)
                .domains(b1 -> b1
                        .domain(domain))
                .enabled(false)
                .name("tenant-with-cert")
                .parameters(b2 -> b2
                        .name("tenantName")
                        .value("myTenant"))
                .customizations(b3 -> b3
                        .certificate(b4 -> b4
                                .arn(certificateArn))) // NOTE: Cert must be in
Us-East-1 and cover the domain provided in this request

        );

        final DistributionTenant distributionTenant =
createResponse.distributionTenant();

        // Then update the Route53 hosted zone to point your domain at the
distribution tenant
        // We fetch the RoutingEndpoint to point to via the default connection
group that was created for your tenant
        final GetConnectionGroupResponse fetchedConnectionGroup =
cloudFrontClient.getConnectionGroup(builder -> builder
                .identifier(distributionTenant.connectionGroupId()));

        route53Client.changeResourceRecordSets(builder -> builder
                .hostedZoneId(hostedZoneId)
                .changeBatch(b1 -> b1
                        .comment("ChangeBatch comment")
                        .changes(b2 -> b2
                                .resourceRecordSet(b3 -> b3
                                        .name(domain)
                                        .type("CNAME")
```

```
                                        .ttl(300L)
                                        .resourceRecords(b4 -> b4

 .value(fetchedConnectionGroup.connectionGroup().routingEndpoint())))
                                .action("CREATE"))
                ));
        return distributionTenant;
    }

}
```

The following example demonstrates how to do so with a CloudFront-hosted managed certificate request. This is ideal if you don't already have traffic towards your domain. In this case, we create a ConnectionGroup to generate a RoutingEndpoint. Then we use that RoutingEndpoint to create DNS records which verify domain ownership and point to CloudFront. CloudFront will then automatically serve a token to validate domain ownership and create a managed certificate.

```
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import
 software.amazon.awssdk.services.cloudfront.model.CreateConnectionGroupResponse;
import
 software.amazon.awssdk.services.cloudfront.model.CreateDistributionTenantResponse;
import software.amazon.awssdk.services.cloudfront.model.DistributionTenant;
import
 software.amazon.awssdk.services.cloudfront.model.GetConnectionGroupResponse;
import software.amazon.awssdk.services.cloudfront.model.ValidationTokenHost;
import software.amazon.awssdk.services.route53.Route53Client;
import software.amazon.awssdk.services.route53.model.RRType;

import java.time.Instant;

public class CreateDistributionTenant {

    public static DistributionTenant
 createDistributionTenantCfHosted(CloudFrontClient cloudFrontClient,

 Route53Client route53Client,
                                                                    String
 distributionId,
```

```
                                                                        String
domain,
                                                                        String
hostedZoneId) throws InterruptedException {
        CreateConnectionGroupResponse createConnectionGroupResponse =
cloudFrontClient.createConnectionGroup(builder -> builder
                .ipv6Enabled(true)
                .name("cf-hosted-connection-group")
                .enabled(true));

        route53Client.changeResourceRecordSets(builder -> builder
                .hostedZoneId(hostedZoneId)
                .changeBatch(b1 -> b1
                        .comment("cf-hosted domain validation record")
                        .changes(b2 -> b2
                                .resourceRecordSet(b3 -> b3
                                        .name(domain)
                                        .type(RRType.CNAME)
                                        .ttl(300L)
                                        .resourceRecords(b4 -> b4

.value(createConnectionGroupResponse.connectionGroup().routingEndpoint())))
                                .action("CREATE"))
                ));

        // Give the R53 record time to propagate, if it isn't being returned by
servers yet, the following call will fail
        Thread.sleep(60000);

        CreateDistributionTenantResponse createResponse =
cloudFrontClient.createDistributionTenant(builder -> builder
                .distributionId(distributionId)
                .domains(b1 -> b1
                        .domain(domain))

.connectionGroupId(createConnectionGroupResponse.connectionGroup().id())
                .enabled(false)
                .name("cf-hosted-tenant")
                .parameters(b2 -> b2
                        .name("tenantName")
                        .value("myTenant"))
                .managedCertificateRequest(b3 -> b3
                        .validationTokenHost(ValidationTokenHost.CLOUDFRONT)
                )
```

```
        );

        return createResponse.distributionTenant();
    }

}
```

The following example demonstrates how to do so with a self-hosted managed certificate request. This is ideal if you have traffic towards your domain and can't tolerate downtime during a migration. At the end of this example, the Tenant will be created in a state awaiting domain validation and DNS setup. Follow steps [here](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/managed-cloudfront-certificates.html#complete-domain-ownership) to complete setup when you are ready to migrate traffic.

```
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import
 software.amazon.awssdk.services.cloudfront.model.CreateConnectionGroupResponse;
import
 software.amazon.awssdk.services.cloudfront.model.CreateDistributionTenantResponse;
import software.amazon.awssdk.services.cloudfront.model.DistributionTenant;
import
 software.amazon.awssdk.services.cloudfront.model.GetConnectionGroupResponse;
import software.amazon.awssdk.services.cloudfront.model.ValidationTokenHost;
import software.amazon.awssdk.services.route53.Route53Client;
import software.amazon.awssdk.services.route53.model.RRType;

import java.time.Instant;

public class CreateDistributionTenant {

    public static DistributionTenant
 createDistributionTenantSelfHosted(CloudFrontClient cloudFrontClient,
                                                                    String
 distributionId,
                                                                    String
 domain) {
        CreateDistributionTenantResponse createResponse =
 cloudFrontClient.createDistributionTenant(builder -> builder
                .distributionId(distributionId)
```

```
                .domains(b1 -> b1
                        .domain(domain))
                .parameters(b2 -> b2
                        .name("tenantName")
                        .value("myTenant"))
                .enabled(false)
                .name("self-hosted-tenant")
                .managedCertificateRequest(b3 -> b3
                        .validationTokenHost(ValidationTokenHost.SELF_HOSTED)
                        .primaryDomainName(domain)
                )
        );

        return createResponse.distributionTenant();
    }

}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateDistribution
  - CreateDistributionTenant

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Delete CloudFront signing resources using AWS SDK

The following code example shows how to delete resources that are used to gain access to restricted content in an Amazon Simple Storage Service (Amazon S3) bucket.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.cloudfront.CloudFrontClient;
import software.amazon.awssdk.services.cloudfront.model.DeleteKeyGroupResponse;
import
 software.amazon.awssdk.services.cloudfront.model.DeleteOriginAccessControlResponse;
import software.amazon.awssdk.services.cloudfront.model.DeletePublicKeyResponse;
import software.amazon.awssdk.services.cloudfront.model.GetKeyGroupResponse;
import
 software.amazon.awssdk.services.cloudfront.model.GetOriginAccessControlResponse;
import software.amazon.awssdk.services.cloudfront.model.GetPublicKeyResponse;

public class DeleteSigningResources {
    private static final Logger logger =
 LoggerFactory.getLogger(DeleteSigningResources.class);

    public static void deleteOriginAccessControl(final CloudFrontClient
 cloudFrontClient,
            final String originAccessControlId) {
        GetOriginAccessControlResponse getResponse = cloudFrontClient
                .getOriginAccessControl(b -> b.id(originAccessControlId));
        DeleteOriginAccessControlResponse deleteResponse =
 cloudFrontClient.deleteOriginAccessControl(builder -> builder
                .id(originAccessControlId)
                .ifMatch(getResponse.eTag()));
        if (deleteResponse.sdkHttpResponse().isSuccessful()) {
            logger.info("Successfully deleted Origin Access Control [{}]",
 originAccessControlId);
        }
    }

    public static void deleteKeyGroup(final CloudFrontClient cloudFrontClient,
 final String keyGroupId) {

        GetKeyGroupResponse getResponse = cloudFrontClient.getKeyGroup(b ->
 b.id(keyGroupId));
        DeleteKeyGroupResponse deleteResponse =
 cloudFrontClient.deleteKeyGroup(builder -> builder
                .id(keyGroupId)
                .ifMatch(getResponse.eTag()));
        if (deleteResponse.sdkHttpResponse().isSuccessful()) {
            logger.info("Successfully deleted Key Group [{}]", keyGroupId);
        }
```

```
    }

    public static void deletePublicKey(final CloudFrontClient cloudFrontClient,
 final String publicKeyId) {
        GetPublicKeyResponse getResponse = cloudFrontClient.getPublicKey(b ->
 b.id(publicKeyId));

        DeletePublicKeyResponse deleteResponse =
 cloudFrontClient.deletePublicKey(builder -> builder
                .id(publicKeyId)
                .ifMatch(getResponse.eTag()));

        if (deleteResponse.sdkHttpResponse().isSuccessful()) {
            logger.info("Successfully deleted Public Key [{}]", publicKeyId);
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - DeleteKeyGroup
  - DeleteOriginAccessControl
  - DeletePublicKey

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get started with a basic content distribution network

The following code example shows how to:

- Create an Amazon S3 bucket
- Upload content to the bucket
- Create a CloudFront distribution with OAC
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# CloudFront Getting Started Tutorial Script
# This script creates an S3 bucket, uploads sample content, creates a CloudFront
 distribution with OAC,
# and demonstrates how to access content through CloudFront.

set -euo pipefail

# Set up logging
LOG_FILE="cloudfront-tutorial.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting CloudFront Getting Started Tutorial at $(date)"

# Function to handle errors
handle_error() {
    echo "ERROR: $1" >&2
    echo "Resources created before error:"
    if [ -n "${BUCKET_NAME:-}" ]; then
        echo "- S3 Bucket: $BUCKET_NAME"
    fi
    if [ -n "${OAC_ID:-}" ]; then
        echo "- CloudFront Origin Access Control: $OAC_ID"
    fi
    if [ -n "${DISTRIBUTION_ID:-}" ]; then
        echo "- CloudFront Distribution: $DISTRIBUTION_ID"
    fi

    echo "Attempting to clean up resources..."
    cleanup
    exit 1
```

```
}

# Function to clean up resources
cleanup() {
    echo "Cleaning up resources..."

    if [ -n "${DISTRIBUTION_ID:-}" ]; then
        echo "Disabling CloudFront distribution $DISTRIBUTION_ID..."

        # Get the current configuration and ETag in one call
        DIST_CONFIG=$(aws cloudfront get-distribution-config --id
 "$DISTRIBUTION_ID" 2>/dev/null) || {
            echo "Failed to get distribution config. Continuing with cleanup..."
            DIST_CONFIG=""
        }

        if [ -n "$DIST_CONFIG" ]; then
            ETAG=$(echo "$DIST_CONFIG" | jq -r '.ETag')

            # Modify and update distribution in one pipeline
            if echo "$DIST_CONFIG" | jq '.DistributionConfig.Enabled = false' | \
                aws cloudfront update-distribution \
                    --id "$DISTRIBUTION_ID" \
                    --distribution-config "$(cat)" \
                    --if-match "$ETAG" 2>/dev/null; then

                echo "Waiting for distribution to be disabled (this may take
 several minutes)..."
                aws cloudfront wait distribution-deployed --id "$DISTRIBUTION_ID"
 2>/dev/null || {
                    echo "Distribution deployment wait timed out. Proceeding with
 deletion..."
                }

                # Get fresh ETag for deletion
                DIST_CONFIG=$(aws cloudfront get-distribution-config --id
 "$DISTRIBUTION_ID" 2>/dev/null) || {
                    echo "Failed to get updated config. Skipping distribution
 deletion..."
                    DIST_CONFIG=""
                }

                if [ -n "$DIST_CONFIG" ]; then
                    ETAG=$(echo "$DIST_CONFIG" | jq -r '.ETag')
```

```
                    aws cloudfront delete-distribution --id "$DISTRIBUTION_ID" --
if-match "$ETAG" 2>/dev/null && \
                        echo "CloudFront distribution deleted." || \
                        echo "Failed to delete distribution. You may need to
 delete it manually."
                fi
            else
                echo "Failed to disable distribution. Continuing with cleanup..."
            fi
        fi
    fi

    if [ -n "${OAC_ID:-}" ]; then
        echo "Deleting Origin Access Control $OAC_ID..."
        OAC_DATA=$(aws cloudfront get-origin-access-control --id "$OAC_ID" 2>/
dev/null) || {
            echo "Failed to get Origin Access Control. You may need to delete it
 manually."
            OAC_DATA=""
        }

        if [ -n "$OAC_DATA" ]; then
            OAC_ETAG=$(echo "$OAC_DATA" | jq -r '.ETag')
            aws cloudfront delete-origin-access-control --id "$OAC_ID" --if-match
 "$OAC_ETAG" 2>/dev/null && \
                echo "Origin Access Control deleted." || \
                echo "Failed to delete Origin Access Control. You may need to
 delete it manually."
        fi
    fi

    if [ -n "${BUCKET_NAME:-}" ] && [ "$BUCKET_IS_SHARED" != "true" ]; then
        echo "Deleting S3 bucket $BUCKET_NAME and its contents..."
        aws s3 rm "s3://$BUCKET_NAME" --recursive 2>/dev/null || {
            echo "Failed to remove bucket contents. Continuing with bucket
 deletion..."
        }

        aws s3 rb "s3://$BUCKET_NAME" 2>/dev/null && \
            echo "S3 bucket deleted." || \
            echo "Failed to delete bucket. You may need to delete it manually."
    fi

    # Clean up temporary files
```

```
    rm -f temp_disabled_config.json distribution-config.json bucket-policy.json
 2>/dev/null || true
    rm -rf temp_content 2>/dev/null || true
}

# Trap errors and cleanup
trap 'handle_error "Script interrupted"' INT TERM

# Validate AWS CLI is available
if ! command -v aws &> /dev/null; then
    handle_error "AWS CLI is not installed or not in PATH"
fi

# Validate jq is available
if ! command -v jq &> /dev/null; then
    handle_error "jq is not installed or not in PATH"
fi

# Validate AWS credentials are configured
if ! aws sts get-caller-identity &> /dev/null; then
    handle_error "AWS credentials are not configured or invalid"
fi

# Initialize variables
BUCKET_NAME=""
OAC_ID=""
DISTRIBUTION_ID=""
BUCKET_IS_SHARED=false

# Generate a random identifier for the bucket name using secure random
RANDOM_ID=$(openssl rand -hex 6)
if [ -z "$RANDOM_ID" ]; then
    handle_error "Failed to generate random identifier"
fi

# Check for shared prereq bucket and get account ID in parallel calls
PREREQ_BUCKET=$(aws cloudformation describe-stacks --stack-name tutorial-prereqs-
bucket \
    --query 'Stacks[0].Outputs[?OutputKey==`BucketName`].OutputValue' --output
 text 2>/dev/null) || PREREQ_BUCKET=""

if [ -n "$PREREQ_BUCKET" ] && [ "$PREREQ_BUCKET" != "None" ]; then
    BUCKET_NAME="$PREREQ_BUCKET"
    BUCKET_IS_SHARED=true
```

```
    echo "Using shared bucket: $BUCKET_NAME"
else
    BUCKET_IS_SHARED=false
    BUCKET_NAME="cloudfront-${RANDOM_ID}"
fi
echo "Using bucket name: $BUCKET_NAME"

# Get AWS account ID early
ACCOUNT_ID=$(aws sts get-caller-identity --query 'Account' --output text)
if [ $? -ne 0 ]; then
    handle_error "Failed to get AWS account ID"
fi

# Validate account ID format
if ! [[ "$ACCOUNT_ID" =~ ^[0-9]{12}$ ]]; then
    handle_error "Invalid AWS account ID format: $ACCOUNT_ID"
fi

# Create a temporary directory for content with restrictive permissions
TEMP_DIR="temp_content"
mkdir -p "$TEMP_DIR/css"
chmod 700 "$TEMP_DIR"
if [ $? -ne 0 ]; then
    handle_error "Failed to create temporary directory"
fi

# Step 1: Create an S3 bucket (only if not shared)
if [ "$BUCKET_IS_SHARED" != "true" ]; then
    echo "Creating S3 bucket: $BUCKET_NAME"
    aws s3 mb "s3://$BUCKET_NAME" --region us-east-1
    if [ $? -ne 0 ]; then
        handle_error "Failed to create S3 bucket"
    fi

    aws s3api put-bucket-tagging --bucket "$BUCKET_NAME" --tagging
 'TagSet=[{Key=project,Value=doc-smith},{Key=tutorial,Value=cloudfront-
gettingstarted}]'

    # Batch bucket configuration calls for efficiency
    aws s3api put-bucket-versioning --bucket "$BUCKET_NAME" --versioning-
configuration Status=Enabled &
    aws s3api put-public-access-block \
        --bucket "$BUCKET_NAME" \
```

```
        --public-access-block-configuration
 "BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets
 &
    aws s3api put-bucket-encryption \
        --bucket "$BUCKET_NAME" \
        --server-side-encryption-configuration '{
            "Rules": [
                {
                    "ApplyServerSideEncryptionByDefault": {
                        "SSEAlgorithm": "AES256"
                    }
                }
            ]
        }' &
    wait
fi

# Step 2: Create sample content
echo "Creating sample content..."
cat > "$TEMP_DIR/index.html" << 'EOF'
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
    <link rel="stylesheet" type="text/css" href="css/styles.css">
</head>
<body>
    <h1>Hello world!</h1>
</body>
</html>
EOF

cat > "$TEMP_DIR/css/styles.css" << 'EOF'
body {
    font-family: Arial, sans-serif;
    margin: 40px;
    background-color: #f5f5f5;
}
h1 {
    color: #333;
    text-align: center;
}
EOF
```

```
# Set restrictive permissions on content files
chmod 600 "$TEMP_DIR/index.html" "$TEMP_DIR/css/styles.css"

# Step 3: Upload content to the S3 bucket with encryption and metadata
echo "Uploading content to S3 bucket..."
aws s3 cp "$TEMP_DIR/" "s3://$BUCKET_NAME/" --recursive \
    --sse AES256 \
    --metadata "Source=CloudFrontTutorial"
if [ $? -ne 0 ]; then
    handle_error "Failed to upload content to S3 bucket"
fi

# Step 4: Create Origin Access Control
echo "Creating Origin Access Control..."
OAC_RESPONSE=$(aws cloudfront create-origin-access-control \
    --origin-access-control-config Name="oac-for-
$BUCKET_NAME",SigningProtocol=sigv4,SigningBehavior=always,OriginAccessControlOriginType=

if [ $? -ne 0 ]; then
    handle_error "Failed to create Origin Access Control"
fi

OAC_ID=$(echo "$OAC_RESPONSE" | jq -r '.OriginAccessControl.Id')
if [ -z "$OAC_ID" ] || [ "$OAC_ID" = "null" ]; then
    handle_error "Failed to extract OAC ID from response"
fi

# Validate OAC ID format (alphanumeric and hyphens)
if ! [[ "$OAC_ID" =~ ^[A-Z0-9]+$ ]]; then
    handle_error "Invalid OAC ID format: $OAC_ID"
fi

echo "Created Origin Access Control with ID: $OAC_ID"

# Step 5: Create CloudFront distribution
echo "Creating CloudFront distribution..."

# Validate bucket name format
if ! [[ "$BUCKET_NAME" =~ ^[a-z0-9][a-z0-9.-]*[a-z0-9]$ ]]; then
    handle_error "Invalid S3 bucket name format: $BUCKET_NAME"
fi

# Create distribution configuration with improved security settings
cat > distribution-config.json << EOF
```

```
{
    "CallerReference": "cli-tutorial-$(date +%s)",
    "Origins": {
        "Quantity": 1,
        "Items": [
            {
                "Id": "S3-$BUCKET_NAME",
                "DomainName": "$BUCKET_NAME.s3.amazonaws.com",
                "S3OriginConfig": {
                    "OriginAccessIdentity": ""
                },
                "OriginAccessControlId": "$OAC_ID"
            }
        ]
    },
    "DefaultCacheBehavior": {
        "TargetOriginId": "S3-$BUCKET_NAME",
        "ViewerProtocolPolicy": "redirect-to-https",
        "AllowedMethods": {
            "Quantity": 2,
            "Items": ["GET", "HEAD"],
            "CachedMethods": {
                "Quantity": 2,
                "Items": ["GET", "HEAD"]
            }
        },
        "DefaultTTL": 86400,
        "MinTTL": 0,
        "MaxTTL": 31536000,
        "Compress": true,
        "ForwardedValues": {
            "QueryString": false,
            "Cookies": {
                "Forward": "none"
            }
        }
    },
    "Comment": "CloudFront distribution for tutorial",
    "Enabled": true,
    "WebACLId": "",
    "HttpVersion": "http2and3"
}
EOF
```

```
# Set restrictive permissions on config file before passing credentials
chmod 600 distribution-config.json

DIST_RESPONSE=$(aws cloudfront create-distribution --distribution-config file://
distribution-config.json)
if [ $? -ne 0 ]; then
    handle_error "Failed to create CloudFront distribution"
fi

DISTRIBUTION_ID=$(echo "$DIST_RESPONSE" | jq -r '.Distribution.Id')
DOMAIN_NAME=$(echo "$DIST_RESPONSE" | jq -r '.Distribution.DomainName')

if [ -z "$DISTRIBUTION_ID" ] || [ "$DISTRIBUTION_ID" = "null" ] || [ -z
 "$DOMAIN_NAME" ] || [ "$DOMAIN_NAME" = "null" ]; then
    handle_error "Failed to extract distribution ID or domain name from response"
fi

# Validate distribution ID format
if ! [[ "$DISTRIBUTION_ID" =~ ^[A-Z0-9]+$ ]]; then
    handle_error "Invalid distribution ID format: $DISTRIBUTION_ID"
fi

echo "Created CloudFront distribution with ID: $DISTRIBUTION_ID"
echo "CloudFront domain name: $DOMAIN_NAME"

# Tag the CloudFront distribution
aws cloudfront tag-resource --resource "arn:aws:cloudfront::
$ACCOUNT_ID:distribution/$DISTRIBUTION_ID" --tags 'Items=[{Key=project,Value=doc-
smith},{Key=tutorial,Value=cloudfront-gettingstarted}]'

# Step 6: Update S3 bucket policy
echo "Updating S3 bucket policy..."

cat > bucket-policy.json << EOF
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontServicePrincipal",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "s3:GetObject",
```

```
            "Resource": "arn:aws:s3:::$BUCKET_NAME/*",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceArn": "arn:aws:cloudfront::
$ACCOUNT_ID:distribution/$DISTRIBUTION_ID"
                }
            }
        }
    ]
}
EOF

# Set restrictive permissions on policy file
chmod 600 bucket-policy.json

aws s3api put-bucket-policy --bucket "$BUCKET_NAME" --policy file://bucket-
policy.json
if [ $? -ne 0 ]; then
    handle_error "Failed to update S3 bucket policy"
fi

# Step 7: Wait for distribution to deploy
echo "Waiting for CloudFront distribution to deploy (this may take 5-10
 minutes)..."
aws cloudfront wait distribution-deployed --id "$DISTRIBUTION_ID" 2>/dev/null ||
 {
    echo "Warning: Distribution deployment wait timed out. The distribution may
 still be deploying."
}

echo "CloudFront distribution is now deployed."

# Step 8: Display access information
echo ""
echo "===== CloudFront Distribution Setup Complete ====="
echo "You can access your content at: https://$DOMAIN_NAME/index.html"
echo ""
echo "Resources created:"
echo "- S3 Bucket: $BUCKET_NAME"
echo "- CloudFront Origin Access Control: $OAC_ID"
echo "- CloudFront Distribution: $DISTRIBUTION_ID"
echo ""
echo "To clean up resources, run: cleanup"
echo ""
```

```
echo "Tutorial completed at $(date)"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - CreateDistribution
  - CreateOriginAccessControl
  - DeleteDistribution
  - DeleteOriginAccessControl
  - GetDistribution
  - GetDistributionConfig
  - GetOriginAccessControl
  - UpdateDistribution
  - WaitDistributionDeployed

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with web application firewalls

The following code example shows how to:

- Create a web ACL
- Add a string match rule
- Add managed rules
- Configure logging
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# AWS WAF Getting Started Script
# This script creates a Web ACL with a string match rule and AWS Managed Rules,
# associates it with a CloudFront distribution, and then cleans up all resources.

set -euo pipefail

# Security: Restrict file permissions
umask 077

# Set up logging with secure file handling
LOG_FILE="waf-tutorial.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "================================================="
echo "AWS WAF Getting Started Tutorial"
echo "================================================="
echo "This script will create AWS WAF resources and associate"
echo "them with a CloudFront distribution."
echo ""

# Maximum number of retries for operations
MAX_RETRIES=3

# Function to handle errors securely
handle_error() {
    local error_msg="$1"
    echo "ERROR: $error_msg" >&2
    echo "Check the log file for details: $LOG_FILE" >&2
```

```
    cleanup_resources
    exit 1
}

# Function to validate AWS CLI JSON output
validate_json() {
    local json_string="$1"
    if ! echo "$json_string" | jq empty 2>/dev/null; then
        return 1
    fi
    return 0
}

# Function to safely extract JSON values
extract_json_value() {
    local json_string="$1"
    local key_path="$2"

    if ! validate_json "$json_string"; then
        return 1
    fi

    echo "$json_string" | jq -r "$key_path" 2>/dev/null || return 1
}

# Function to clean up resources securely
cleanup_resources() {
    echo ""
    echo "=================================================="
    echo "CLEANING UP RESOURCES"
    echo "=================================================="

    if [ -n "${DISTRIBUTION_ID:-}" ] && [ -n "${WEB_ACL_ARN:-}" ]; then
        echo "Disassociating Web ACL from CloudFront distribution..."
        local account_id
        account_id=$(aws sts get-caller-identity --query Account --output text
 2>/dev/null) || account_id=""

        if [ -z "$account_id" ]; then
            echo "Warning: Could not retrieve AWS account ID"
            return
        fi

        local disassociate_result
```

```
        disassociate_result=$(aws wafv2 disassociate-web-acl \
            --resource-arn "arn:aws:cloudfront::${account_id}:distribution/
${DISTRIBUTION_ID}" \
            --region us-east-1 2>&1) || true

        if echo "$disassociate_result" | grep -qi "error"; then
            echo "Warning: Failed to disassociate Web ACL: $disassociate_result"
        else
            echo "Web ACL disassociated successfully."
        fi
    fi

    if [ -n "${WEB_ACL_ID:-}" ] && [ -n "${WEB_ACL_NAME:-}" ]; then
        echo "Deleting Web ACL..."

        local get_result
        get_result=$(aws wafv2 get-web-acl \
            --name "$WEB_ACL_NAME" \
            --scope CLOUDFRONT \
            --id "$WEB_ACL_ID" \
            --region us-east-1 2>&1) || true

        if echo "$get_result" | grep -qi "error"; then
            echo "Warning: Failed to get Web ACL for deletion: $get_result"
            echo "You may need to manually delete the Web ACL using the AWS
 Console."
        else
            local latest_token
            latest_token=$(extract_json_value "$get_result" '.WebACL.LockToken'
 2>/dev/null) || latest_token=""

            if [ -n "$latest_token" ]; then
                local delete_result
                delete_result=$(aws wafv2 delete-web-acl \
                    --name "$WEB_ACL_NAME" \
                    --scope CLOUDFRONT \
                    --id "$WEB_ACL_ID" \
                    --lock-token "$latest_token" \
                    --region us-east-1 2>&1) || true

                if echo "$delete_result" | grep -qi "error"; then
                    echo "Warning: Failed to delete Web ACL: $delete_result"
                    echo "You may need to manually delete the Web ACL using the
 AWS Console."
```

```
                else
                    echo "Web ACL deleted successfully."
                fi
            else
                echo "Warning: Could not extract lock token for deletion. You may
 need to manually delete the Web ACL."
            fi
        fi
    fi

    echo "Cleanup process completed."
}

# Security: Trap EXIT to ensure cleanup on any exit
trap cleanup_resources EXIT

# Generate a random identifier for resource names using secure method
RANDOM_ID=$(openssl rand -hex 4) || handle_error "Failed to generate random ID"
WEB_ACL_NAME="MyWebACL-${RANDOM_ID}"
METRIC_NAME="MyWebACLMetrics-${RANDOM_ID}"

echo "Using Web ACL name: $WEB_ACL_NAME"

# Step 1: Create a Web ACL
echo ""
echo "=================================================="
echo "STEP 1: Creating Web ACL"
echo "=================================================="

local create_result
create_result=$(aws wafv2 create-web-acl \
    --name "$WEB_ACL_NAME" \
    --scope "CLOUDFRONT" \
    --default-action Allow={} \
    --visibility-config
 "SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=
$METRIC_NAME" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=aws-waf-gs \
    --region us-east-1 2>&1) || handle_error "Failed to create Web ACL"

if ! validate_json "$create_result"; then
    handle_error "Invalid JSON response from create-web-acl"
fi
```

```
# Extract Web ACL ID, ARN, and Lock Token from the response
WEB_ACL_ID=$(extract_json_value "$create_result" '.Summary.Id') || handle_error
 "Failed to extract Web ACL ID"
WEB_ACL_ARN=$(extract_json_value "$create_result" '.Summary.ARN') || handle_error
 "Failed to extract Web ACL ARN"
LOCK_TOKEN=$(extract_json_value "$create_result" '.Summary.LockToken') ||
 handle_error "Failed to extract Lock Token"

echo "Web ACL created successfully with ID: $WEB_ACL_ID"
echo "Lock Token: [REDACTED]"

# Step 2: Add a String Match Rule
echo ""
echo "=================================================="
echo "STEP 2: Adding String Match Rule"
echo "=================================================="

for ((i=1; i<=MAX_RETRIES; i++)); do
    echo "Attempt $i to add string match rule..."

    # Get the latest lock token before updating
    local get_result
    get_result=$(aws wafv2 get-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope CLOUDFRONT \
        --id "$WEB_ACL_ID" \
        --region us-east-1 2>&1) || true

    if echo "$get_result" | grep -qi "error"; then
        echo "Warning: Failed to get Web ACL for update: $get_result"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to get Web ACL after $MAX_RETRIES attempts"
        fi
        sleep 2
        continue
    fi

    if ! validate_json "$get_result"; then
        echo "Warning: Invalid JSON response from get-web-acl"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Invalid JSON response after $MAX_RETRIES attempts"
        fi
        sleep 2
        continue
```

```
    fi

    local latest_token
    latest_token=$(extract_json_value "$get_result" '.WebACL.LockToken' 2>/dev/
null) || true

    if [ -z "$latest_token" ]; then
        echo "Warning: Could not extract lock token for update"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to extract lock token after $MAX_RETRIES
 attempts"
        fi
        sleep 2
        continue
    fi

    local update_result
    update_result=$(aws wafv2 update-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope "CLOUDFRONT" \
        --id "$WEB_ACL_ID" \
        --lock-token "$latest_token" \
        --default-action Allow={} \
        --rules '[{
            "Name": "UserAgentRule",
            "Priority": 0,
            "Statement": {
                "ByteMatchStatement": {
                    "SearchString": "MyAgent",
                    "FieldToMatch": {
                        "SingleHeader": {
                            "Name": "user-agent"
                        }
                    },
                    "TextTransformations": [
                        {
                            "Priority": 0,
                            "Type": "NONE"
                        }
                    ],
                    "PositionalConstraint": "EXACTLY"
                }
            },
            "Action": {
```

```
                "Count": {}
            },
            "VisibilityConfig": {
                "SampledRequestsEnabled": true,
                "CloudWatchMetricsEnabled": true,
                "MetricName": "UserAgentRuleMetric"
            }
        }]' \
        --visibility-config
 "SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=
$METRIC_NAME" \
        --region us-east-1 2>&1) || true

    if echo "$update_result" | grep -qi "WAFOptimisticLockException"; then
        echo "Optimistic lock exception encountered. Will retry with new lock
 token."
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to add string match rule after $MAX_RETRIES
 attempts"
        fi
        sleep 2
        continue
    elif echo "$update_result" | grep -qi "error"; then
        handle_error "Failed to add string match rule: $update_result"
    else
        echo "String match rule added successfully."
        break
    fi
done

# Step 3: Add AWS Managed Rules
echo ""
echo "=================================================="
echo "STEP 3: Adding AWS Managed Rules"
echo "=================================================="

for ((i=1; i<=MAX_RETRIES; i++)); do
    echo "Attempt $i to add AWS Managed Rules..."

    # Get the latest lock token before updating
    local get_result
    get_result=$(aws wafv2 get-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope CLOUDFRONT \
```

```
        --id "$WEB_ACL_ID" \
        --region us-east-1 2>&1) || true

    if echo "$get_result" | grep -qi "error"; then
        echo "Warning: Failed to get Web ACL for update: $get_result"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to get Web ACL after $MAX_RETRIES attempts"
        fi
        sleep 2
        continue
    fi

    if ! validate_json "$get_result"; then
        echo "Warning: Invalid JSON response from get-web-acl"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Invalid JSON response after $MAX_RETRIES attempts"
        fi
        sleep 2
        continue
    fi

    local latest_token
    latest_token=$(extract_json_value "$get_result" '.WebACL.LockToken' 2>/dev/
null) || true

    if [ -z "$latest_token" ]; then
        echo "Warning: Could not extract lock token for update"
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to extract lock token after $MAX_RETRIES
 attempts"
        fi
        sleep 2
        continue
    fi

    local update_result
    update_result=$(aws wafv2 update-web-acl \
        --name "$WEB_ACL_NAME" \
        --scope "CLOUDFRONT" \
        --id "$WEB_ACL_ID" \
        --lock-token "$latest_token" \
        --default-action Allow={} \
        --rules '[{
            "Name": "UserAgentRule",
```

```
            "Priority": 0,
            "Statement": {
                "ByteMatchStatement": {
                    "SearchString": "MyAgent",
                    "FieldToMatch": {
                        "SingleHeader": {
                            "Name": "user-agent"
                        }
                    },
                    "TextTransformations": [
                        {
                            "Priority": 0,
                            "Type": "NONE"
                        }
                    ],
                    "PositionalConstraint": "EXACTLY"
                }
            },
            "Action": {
                "Count": {}
            },
            "VisibilityConfig": {
                "SampledRequestsEnabled": true,
                "CloudWatchMetricsEnabled": true,
                "MetricName": "UserAgentRuleMetric"
            }
        },
        {
            "Name": "AWS-AWSManagedRulesCommonRuleSet",
            "Priority": 1,
            "Statement": {
                "ManagedRuleGroupStatement": {
                    "VendorName": "AWS",
                    "Name": "AWSManagedRulesCommonRuleSet",
                    "ExcludedRules": []
                }
            },
            "OverrideAction": {
                "Count": {}
            },
            "VisibilityConfig": {
                "SampledRequestsEnabled": true,
                "CloudWatchMetricsEnabled": true,
                "MetricName": "AWS-AWSManagedRulesCommonRuleSet"
```

```
            }
        }]' \
        --visibility-config
 "SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=
$METRIC_NAME" \
        --region us-east-1 2>&1) || true

    if echo "$update_result" | grep -qi "WAFOptimisticLockException"; then
        echo "Optimistic lock exception encountered. Will retry with new lock
 token."
        if [ "$i" -eq "$MAX_RETRIES" ]; then
            handle_error "Failed to add AWS Managed Rules after $MAX_RETRIES
 attempts"
        fi
        sleep 2
        continue
    elif echo "$update_result" | grep -qi "error"; then
        handle_error "Failed to add AWS Managed Rules: $update_result"
    else
        echo "AWS Managed Rules added successfully."
        break
    fi
done

# Step 4: List CloudFront distributions
echo ""
echo "=================================================="
echo "STEP 4: Listing CloudFront Distributions"
echo "=================================================="

local cf_result
cf_result=$(aws cloudfront list-distributions --query "DistributionList.Items[*].
{Id:Id,DomainName:DomainName}" --output table 2>&1) || cf_result=""

if echo "$cf_result" | grep -qi "error"; then
    echo "Warning: Failed to list CloudFront distributions: $cf_result"
    echo "Continuing without CloudFront association."
    DISTRIBUTION_ID=""
else
    echo "$cf_result"

    # Auto-select first CloudFront distribution if available
    echo ""
    echo "=================================================="
```

```
    echo "STEP 5: Associate Web ACL with CloudFront Distribution"
    echo "================================================="

    local first_dist
    first_dist=$(aws cloudfront list-distributions --query
 "DistributionList.Items[0].Id" --output text 2>&1) || first_dist=""

    if [ -n "$first_dist" ] && [ "$first_dist" != "None" ] && ! echo
 "$first_dist" | grep -qi "error"; then
        DISTRIBUTION_ID="$first_dist"
        echo "Using CloudFront distribution: $DISTRIBUTION_ID"

        local account_id
        account_id=$(aws sts get-caller-identity --query Account --output text
 2>/dev/null) || account_id=""

        if [ -z "$account_id" ]; then
            echo "Warning: Could not retrieve AWS account ID for association"
            DISTRIBUTION_ID=""
        else
            local associate_result
            associate_result=$(aws wafv2 associate-web-acl \
                --web-acl-arn "$WEB_ACL_ARN" \
                --resource-arn "arn:aws:cloudfront::${account_id}:distribution/
${DISTRIBUTION_ID}" \
                --region us-east-1 2>&1) || true

            if echo "$associate_result" | grep -qi "error"; then
                echo "Warning: Failed to associate Web ACL with CloudFront
 distribution: $associate_result"
                echo "Continuing without CloudFront association."
                DISTRIBUTION_ID=""
            else
                echo "Web ACL associated with CloudFront distribution
 successfully."
            fi
        fi
    else
        echo "No CloudFront distributions available. Skipping association."
        DISTRIBUTION_ID=""
    fi
fi

# Display summary of created resources
```

```
echo ""
echo "=================================================="
echo "RESOURCE SUMMARY"
echo "=================================================="
echo "Web ACL Name: $WEB_ACL_NAME"
echo "Web ACL ID: $WEB_ACL_ID"
echo "Web ACL ARN: $WEB_ACL_ARN"
if [ -n "${DISTRIBUTION_ID:-}" ]; then
    echo "Associated CloudFront Distribution: $DISTRIBUTION_ID"
fi
echo ""

# Auto-confirm cleanup
echo "=================================================="
echo "CLEANUP CONFIRMATION"
echo "=================================================="
echo "Proceeding with automatic cleanup of all created resources..."

echo ""
echo "=================================================="
echo "Tutorial completed!"
echo "=================================================="
echo "Log file: $LOG_FILE"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AssociateWebAcl
  - CreateWebAcl
  - DeleteWebAcl
  - DisassociateWebAcl
  - GetCallerIdentity
  - GetWebAcl
  - ListDistributions
  - UpdateWebAcl

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create signed URLs and cookies using an AWS SDK

The following code example shows how to create signed URLs and cookies that allow access to restricted resources.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Use the CannedSignerRequest class to sign URLs or cookies with a *canned* policy.

```
import software.amazon.awssdk.services.cloudfront.model.CannedSignerRequest;

import java.net.URL;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Instant;
import java.time.temporal.ChronoUnit;

public class CreateCannedPolicyRequest {

    public static CannedSignerRequest createRequestForCannedPolicy(String
 distributionDomainName,
            String fileNameToUpload,
            String privateKeyFullPath, String publicKeyId) throws Exception {
        String protocol = "https";
        String resourcePath = "/" + fileNameToUpload;

        String cloudFrontUrl = new URL(protocol, distributionDomainName,
 resourcePath).toString();
        Instant expirationDate = Instant.now().plus(7, ChronoUnit.DAYS);
        Path path = Paths.get(privateKeyFullPath);

        return CannedSignerRequest.builder()
                .resourceUrl(cloudFrontUrl)
                .privateKey(path)
```

```
                .keyPairId(publicKeyId)
                .expirationDate(expirationDate)
                .build();
    }
}
```

Use the CustomSignerRequest class to sign URLs or cookies with a *custom* policy. The `activeDate` and `ipRange` are optional methods.

```
import software.amazon.awssdk.services.cloudfront.model.CustomSignerRequest;

import java.net.URL;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Instant;
import java.time.temporal.ChronoUnit;

public class CreateCustomPolicyRequest {

    public static CustomSignerRequest createRequestForCustomPolicy(String
 distributionDomainName,
            String fileNameToUpload,
            String privateKeyFullPath, String publicKeyId) throws Exception {
        String protocol = "https";
        String resourcePath = "/" + fileNameToUpload;

        String cloudFrontUrl = new URL(protocol, distributionDomainName,
 resourcePath).toString();
        Instant expireDate = Instant.now().plus(7, ChronoUnit.DAYS);
        // URL will be accessible tomorrow using the signed URL.
        Instant activeDate = Instant.now().plus(1, ChronoUnit.DAYS);
        Path path = Paths.get(privateKeyFullPath);

        return CustomSignerRequest.builder()
                .resourceUrl(cloudFrontUrl)
                // .resourceUrlPattern("https://*.example.com/*")  // Optional.
                .privateKey(path)
                .keyPairId(publicKeyId)
                .expirationDate(expireDate)
                .activeDate(activeDate) // Optional.
                // .ipRange("192.168.0.1/24") // Optional.
                .build();
```

```
    }
}
```

The following example demonstrates the use of the CloudFrontUtilities class to produce signed cookies and URLs. View this code example on GitHub.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.cloudfront.CloudFrontUtilities;
import software.amazon.awssdk.services.cloudfront.cookie.CookiesForCannedPolicy;
import software.amazon.awssdk.services.cloudfront.cookie.CookiesForCustomPolicy;
import software.amazon.awssdk.services.cloudfront.model.CannedSignerRequest;
import software.amazon.awssdk.services.cloudfront.model.CustomSignerRequest;
import software.amazon.awssdk.services.cloudfront.url.SignedUrl;

public class SigningUtilities {
    private static final Logger logger =
 LoggerFactory.getLogger(SigningUtilities.class);
    private static final CloudFrontUtilities cloudFrontUtilities =
 CloudFrontUtilities.create();

    public static SignedUrl signUrlForCannedPolicy(CannedSignerRequest
 cannedSignerRequest) {
        SignedUrl signedUrl =
 cloudFrontUtilities.getSignedUrlWithCannedPolicy(cannedSignerRequest);
        logger.info("Signed URL: [{}]", signedUrl.url());
        return signedUrl;
    }

    public static SignedUrl signUrlForCustomPolicy(CustomSignerRequest
 customSignerRequest) {
        SignedUrl signedUrl =
 cloudFrontUtilities.getSignedUrlWithCustomPolicy(customSignerRequest);
        logger.info("Signed URL: [{}]", signedUrl.url());
        return signedUrl;
    }

    public static CookiesForCannedPolicy
 getCookiesForCannedPolicy(CannedSignerRequest cannedSignerRequest) {
        CookiesForCannedPolicy cookiesForCannedPolicy = cloudFrontUtilities
                .getCookiesForCannedPolicy(cannedSignerRequest);
```

```
        logger.info("Cookie EXPIRES header [{}]",
 cookiesForCannedPolicy.expiresHeaderValue());
        logger.info("Cookie KEYPAIR header [{}]",
 cookiesForCannedPolicy.keyPairIdHeaderValue());
        logger.info("Cookie SIGNATURE header [{}]",
 cookiesForCannedPolicy.signatureHeaderValue());
        return cookiesForCannedPolicy;
    }

    public static CookiesForCustomPolicy
 getCookiesForCustomPolicy(CustomSignerRequest customSignerRequest) {
        CookiesForCustomPolicy cookiesForCustomPolicy = cloudFrontUtilities
                .getCookiesForCustomPolicy(customSignerRequest);
        logger.info("Cookie POLICY header [{}]",
 cookiesForCustomPolicy.policyHeaderValue());
        logger.info("Cookie KEYPAIR header [{}]",
 cookiesForCustomPolicy.keyPairIdHeaderValue());
        logger.info("Cookie SIGNATURE header [{}]",
 cookiesForCustomPolicy.signatureHeaderValue());
        return cookiesForCustomPolicy;
    }
}
```

- For API details, see CloudFrontUtilities in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

CloudFront Functions examples for CloudFront

The following code examples show how to use CloudFront with AWS SDKs.

Examples

- Add HTTP security headers to a CloudFront Functions viewer response event
- Add a CORS header to a CloudFront Functions viewer response event
- Add a cache control header to a CloudFront Functions viewer response event
- Add a true client IP header to a CloudFront Functions viewer request event

- Add an origin header to a CloudFront Functions viewer request event
- Add index.html to request URLs without a file name in a CloudFront Functions viewer request event
- Authorize each request before the cache with a network call in a CloudFront Functions viewer request event
- Authorize a request with a network call in a CloudFront Functions viewer request event
- Check for bots with a network call that fails open in a CloudFront Functions viewer request event
- Fail over to a secondary origin after a health check network call in a CloudFront Functions viewer request event
- Read configuration through KeyValueStore with a network call in a CloudFront Functions viewer request event
- Make a network call to an origin from a CloudFront Functions viewer request event
- Serve markdown to AI agents and queue conversions with a network call from CloudFront Functions
- Normalize query string parameters in a CloudFront Functions viewer request
- Redirect to a new URL in a CloudFront Functions viewer request event
- Rewrite a request URI based on KeyValueStore configuration for a CloudFront Functions viewer request event
- Route a request through a private service behind a VPC origin with a network call in a CloudFront Functions viewer request event
- Route requests to an origin closer to the viewer in a CloudFront Functions viewer request event
- Send fire-and-forget events with a network call from CloudFront Functions
- Assign a sticky A/B variant with KeyValueStore and a network call in CloudFront Functions
- Use key-value pairs in a CloudFront Functions viewer request
- Validate an Amazon Cognito access token with a network call in a CloudFront Functions viewer request event
- Validate a simple token in a CloudFront Functions viewer request
- Run a virtual waiting room with a network call in a CloudFront Functions viewer request event

Add HTTP security headers to a CloudFront Functions viewer response event

The following code example shows how to add HTTP security headers to a CloudFront Functions viewer response event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
async function handler(event) {
    var response = event.response;
    var headers = response.headers;

    // Set HTTP security headers
    // Since JavaScript doesn't allow for hyphens in variable names, we use the
 dict["key"] notation
    headers['strict-transport-security'] = { value: 'max-age=63072000;
 includeSubdomains; preload'};
    headers['content-security-policy'] = { value: "default-src 'none'; img-src
 'self'; script-src 'self'; style-src 'self'; object-src 'none'; frame-ancestors
 'none'"};
    headers['x-content-type-options'] = { value: 'nosniff'};
    headers['x-frame-options'] = {value: 'DENY'};
    headers['x-xss-protection'] = {value: '1; mode=block'};
    headers['referrer-policy'] = {value: 'same-origin'};

    // Return the response to viewers
    return response;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Add a CORS header to a CloudFront Functions viewer response event

The following code example shows how to add a CORS header to a CloudFront Functions viewer response event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
async function handler(event)  {
    var request = event.request;
    var response  = event.response;

    // If Access-Control-Allow-Origin CORS header is missing, add it.
    // Since JavaScript doesn't allow for hyphens in variable names, we use the
 dict["key"] notation.
    if (!response.headers['access-control-allow-origin'] &&
 request.headers['origin']) {
        response.headers['access-control-allow-origin'] = {value:
 request.headers['origin'].value};
        console.log("Access-Control-Allow-Origin was missing, adding it now.");
    }

    return response;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Add a cache control header to a CloudFront Functions viewer response event

The following code example shows how to add a cache control header to a CloudFront Functions viewer response event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
async function handler(event) {
    var response = event.response;
    var headers = response.headers;

    if (response.statusCode >= 200 && response.statusCode < 400) {
        // Set the cache-control header
        headers['cache-control'] = {value: 'public, max-age=63072000'};
    }

    // Return response to viewers
    return response;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Add a true client IP header to a CloudFront Functions viewer request event

The following code example shows how to add a true client IP header to a CloudFront Functions viewer request event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
async function handler(event) {
    var request = event.request;
    var clientIP = event.viewer.ip;

    //Add the true-client-ip header to the incoming request
    request.headers['true-client-ip'] = {value: clientIP};

    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Add an origin header to a CloudFront Functions viewer request event

The following code example shows how to add an origin header to a CloudFront Functions viewer request event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
async function handler(event) {
    var request = event.request;
    var headers = request.headers;
    var host = request.headers.host.value;

   // If origin header is missing, set it equal to the host header.
   if (!headers.origin)
       headers.origin = {value:`https://${host}`};

   return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Add index.html to request URLs without a file name in a CloudFront Functions viewer request event

The following code example shows how to add index.html to request URLs without a file name in a CloudFront Functions viewer request event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
async function handler(event) {
    var request = event.request;
    var uri = request.uri;

    // Check whether the URI is missing a file name.
    if (uri.endsWith('/')) {
        request.uri += 'index.html';
    }
    // Check whether the URI is missing a file extension.
    else if (!uri.includes('.')) {
        request.uri += '/index.html';
    }

    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Authorize each request before the cache with a network call in a CloudFront Functions viewer request event

The following code example shows how to ask an authorization origin with cf.fetch whether a request may act on a resource, before the cache serves it.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// A client (for example an AI agent) sends a scoped token on every request. The
 token proves
// who the client is, not whether it may touch THIS resource right now, so the
 function asks an
// authorization service with one network call, POST /v1/check { action,
 resource, token }, and
// forwards the request only on { allowed: true }.
//   No Authorization header -> 401, no call.
//   Not allowed             -> 403.
//   Service cannot be asked -> 503 (fail closed).
// Viewer request runs before the cache, so allowed reads can still be cache
 hits.

// The origin id (the origin's Name in the console) of the authorization service
 on this distribution.
const AUTHZ_ORIGIN_ID = 'authz';
const CHECK_PATH = '/v1/check';
const ERROR_HEADER = 'x-agent-authz-error';
const STATUS_TEXT = { 401: 'Unauthorized', 403: 'Forbidden', 503: 'Service
 Unavailable' };

// What the client wants to do: read or write this URI, with this token.
function accessCheck(request, token) {
    return {
        action: request.method === 'GET' || request.method === 'HEAD' ? 'read' :
 'write',
        resource: request.uri,
        // In the body, not the call's authorization header: with origin access
 control,
```

```
        // CloudFront signs the call and owns that header.
        token: token,
    };
}

// The one network call. Resolves to { ok, status, decision }; throws when the
 service
// cannot be asked or its answer is not JSON.
async function askAuthz(check) {
    const body = JSON.stringify(check);
    const authzResponse = await cf.fetch(CHECK_PATH, {
        originId: AUTHZ_ORIGIN_ID,
        method: 'POST',
        headers: { 'content-type': 'application/json' }, // plain strings, not
 { value } objects
        body: body,
        includeBody: true, // default false; the decision is in the body
    });
    if (!authzResponse.ok) {
        return { ok: false, status: authzResponse.status };
    }
    const text = await authzResponse.text(); // assign first: no await inside
 call arguments
    return { ok: true, status: authzResponse.status, decision:
 JSON.parse(text) };
}

function deny(status, code, error) {
    const headers = {
        'content-type': { value: 'application/json' },
        'cache-control': { value: 'no-store' },
    };
    if (status === 401) {
        headers['www-authenticate'] = { value: 'Bearer' };
    }
    if (error) {
        headers[ERROR_HEADER] = { value: error };
    }
    return {
        statusCode: status,
        statusDescription: STATUS_TEXT[status],
        headers: headers,
        body: JSON.stringify({ error: code }),
    };
```

```
}

async function handler(event) {
    const request = event.request;
    const authorization = request.headers.authorization;
    if (!authorization) {
        return deny(401, 'unauthorized');
    }

    let answer;
    try {
        answer = await askAuthz(accessCheck(request, authorization.value));
    } catch (error) {
        // Fail closed: TypeError, TimeoutError, InternalError, or a body that is
 not JSON.
        return deny(503, 'authz_unavailable', error.name + ': ' + error.message);
    }
    if (!answer.ok) {
        return deny(503, 'authz_unavailable', 'authz returned ' + answer.status);
    }

    const decision = answer.decision;
    if (!decision || decision.allowed !== true) {
        return deny(403, 'forbidden');
    }
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Authorize a request with a network call in a CloudFront Functions viewer request event

The following code example shows how to ask an authorizer origin with cf.fetch whether to allow a request, and forward the identity headers it returns.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// A Lambda authorizer with context, at the edge: one network call POSTs the
 viewer context
// to an authorizer on this distribution, the authorizer answers { allow,
 headers }, and the
// function copies the allowed identity headers onto the request (allow) or
 answers 403 (deny).
// An authorizer that cannot be asked fails closed with 503.

// The origin id (the origin's Name in the console) of the authorizer on this
 distribution.
const AUTHORIZER_ORIGIN_ID = 'authorizer';
const AUTHORIZE_PATH = '/authorize';
// The only headers the authorizer may set. Viewer-supplied copies are always
 deleted.
const IDENTITY_HEADERS = ['x-tenant', 'x-role'];
// The viewer headers sent to the authorizer.
const FORWARDED_HEADERS = ['authorization', 'user-agent'];
const ERROR_HEADER = 'x-authorizer-error';
const STATUS_TEXT = { 401: 'Unauthorized', 403: 'Forbidden', 503: 'Service
 Unavailable' };

function headerValue(headers, name) {
    return headers[name] ? headers[name].value : '';
}

// Never trust identity headers coming from the viewer.
function stripIdentityHeaders(headers) {
    for (let i = 0; i < IDENTITY_HEADERS.length; i++) {
        delete headers[IDENTITY_HEADERS[i]];
```

```
    }
}

// What the authorizer decides on: who is asking, for what.
function viewerContext(event) {
    const request = event.request;
    const headers = {};
    for (let i = 0; i < FORWARDED_HEADERS.length; i++) {
        headers[FORWARDED_HEADERS[i]] = headerValue(request.headers,
 FORWARDED_HEADERS[i]);
    }
    return {
        ip: event.viewer.ip,
        uri: request.uri,
        method: request.method,
        headers: headers,
    };
}

// The one network call. Resolves to { ok, status, decision }; throws when the
 authorizer
// cannot be asked or its answer is not JSON.
async function askAuthorizer(context) {
    const body = JSON.stringify(context);
    const authorizerResponse = await cf.fetch(AUTHORIZE_PATH, {
        originId: AUTHORIZER_ORIGIN_ID,
        method: 'POST',
        headers: { 'content-type': 'application/json' }, // plain strings, not
 { value } objects
        body: body,
        includeBody: true, // default false; the decision is in the body
    });
    if (!authorizerResponse.ok) {
        return { ok: false, status: authorizerResponse.status };
    }
    const text = await authorizerResponse.text(); // assign first: no await
 inside call arguments
    return { ok: true, status: authorizerResponse.status, decision:
 JSON.parse(text) };
}

// Copy only the allowed identity headers, lowercased.
function grantIdentityHeaders(headers, granted) {
    for (const name in granted) {
```

```
        const lowerName = name.toLowerCase();
        if (IDENTITY_HEADERS.indexOf(lowerName) !== -1) {
            headers[lowerName] = { value: String(granted[name]) };
        }
    }
}

function deny(status, code, error) {
    const headers = {
        'content-type': { value: 'application/json' },
        'cache-control': { value: 'no-store' },
    };
    if (error) {
        headers[ERROR_HEADER] = { value: error };
    }
    return {
        statusCode: status,
        statusDescription: STATUS_TEXT[status],
        headers: headers,
        body: JSON.stringify({ error: code }),
    };
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    stripIdentityHeaders(headers);

    // No token, or an empty one: refuse here, without spending the network call.
    const authorization = headers.authorization;
    if (!authorization || !authorization.value.trim()) {
        const missing = deny(401, 'missing_token');
        missing.headers['www-authenticate'] = { value: 'Bearer' };
        return missing;
    }

    let answer;
    try {
        answer = await askAuthorizer(viewerContext(event));
    } catch (error) {
        // Fail closed: TypeError, TimeoutError, InternalError, or a body that is
 not JSON.
        return deny(503, 'authorizer_unavailable', error.name + ': ' +
 error.message);
```

```
    }
    if (!answer.ok) {
        return deny(503, 'authorizer_unavailable', 'authorizer returned ' +
 answer.status);
    }

    const decision = answer.decision;
    if (!decision || decision.allow !== true) {
        return deny(403, 'forbidden');
    }

    grantIdentityHeaders(headers, decision.headers || {});
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Check for bots with a network call that fails open in a CloudFront Functions viewer request event

The following code example shows how to ask a bot-check origin with cf.fetch, combine the verdict with a KeyValueStore policy, and fail open when the call fails.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// Two lookups run in parallel:
```

```
//   1. One network call: POST /check on a bot-detection service on this
 distribution.
//      The only latency limit is the runtime's fixed 1 s timeout.
//   2. A KeyValueStore read of the key bot-policy: block | tag | off. No KVS or
 no key -> tag.
// Service slow or broken -> forward with x-bot-check: unavailable (fail open).
// Verdict bot + block    -> 403 block page.
// Verdict bot + tag      -> forward with x-bot-verdict: bot; the origin decides.
// Policy off             -> forward, verdict ignored.

// The origin id (the origin's Name in the console) of the bot-detection service
 on this distribution.
const VENDOR_ORIGIN_ID = 'bot-check';
const CHECK_PATH = '/check';
const BOT_POLICY_KEY = 'bot-policy';
const POLICIES = ['block', 'tag', 'off'];
const DEFAULT_POLICY = 'tag';
// Every header this function sets starts with this prefix; viewer-supplied ones
 are deleted.
const BOT_HEADER_PREFIX = 'x-bot-';
const POLICY_HEADER = 'x-bot-policy';
const POLICY_SOURCE_HEADER = 'x-bot-policy-source';
const CHECK_HEADER = 'x-bot-check';
const VERDICT_HEADER = 'x-bot-verdict';
const SCORE_HEADER = 'x-bot-score';
const ERROR_HEADER = 'x-bot-check-error';

function headerValue(request, name) {
    return request.headers[name] ? request.headers[name].value : '';
}

function describeError(error) {
    return error.name + ': ' + error.message;
}

// Never trust verdict headers coming from the viewer.
function stripBotHeaders(headers) {
    for (const name in headers) {
        if (name.indexOf(BOT_HEADER_PREFIX) === 0) {
            delete headers[name];
        }
    }
}
```

```
// The one network call. Resolves to { verdict, score } or throws.
async function askVendor(request, viewer) {
    const body = JSON.stringify({
        path: request.uri,
        ip: viewer.ip,
        userAgent: headerValue(request, 'user-agent'),
        acceptLanguage: headerValue(request, 'accept-language'),
    });
    const vendorResponse = await cf.fetch(CHECK_PATH, {
        originId: VENDOR_ORIGIN_ID,
        method: 'POST',
        headers: { 'content-type': 'application/json' }, // plain strings, not
 { value } objects
        body: body,
        includeBody: true, // default false; the verdict is in the body
    });
    if (!vendorResponse.ok) {
        throw new Error('vendor returned ' + vendorResponse.status);
    }
    const text = await vendorResponse.text(); // assign first: no await inside
 call arguments
    const answer = JSON.parse(text);
    if (answer.verdict !== 'bot' && answer.verdict !== 'human') {
        throw new Error('vendor verdict unreadable');
    }
    return { verdict: answer.verdict, score: Number(answer.score) || 0 };
}

// Rejects when no KVS is associated, the key is missing or the value is unknown.
async function readPolicy() {
    const value = await cf.kvs().get(BOT_POLICY_KEY);
    const policy = String(value).trim().toLowerCase();
    if (POLICIES.indexOf(policy) === -1) {
        throw new Error(BOT_POLICY_KEY + ' "' + policy + '" is not block|tag|
off');
    }
    return policy;
}

function blockPage(check) {
    const headers = {
        'content-type': { value: 'text/html; charset=utf-8' },
        'cache-control': { value: 'no-store' },
    };
```

```
    headers[VERDICT_HEADER] = { value: 'bot' };
    headers[SCORE_HEADER] = { value: String(check.score) };
    return {
        statusCode: 403,
        statusDescription: 'Forbidden',
        headers: headers,
        body: '<!doctype html><title>Blocked</title><h1>403 Blocked</h1>',
    };
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    stripBotHeaders(headers);

    // allSettled, not all: a missing KVS key must not throw away the vendor
 verdict.
    const results = await Promise.allSettled([askVendor(request, event.viewer),
 readPolicy()]);
    const vendorResult = results[0];
    const policyResult = results[1];

    const fromKvs = policyResult.status === 'fulfilled';
    const policy = fromKvs ? policyResult.value : DEFAULT_POLICY;
    headers[POLICY_HEADER] = { value: policy };
    headers[POLICY_SOURCE_HEADER] = { value: fromKvs ? 'kvs' : 'default: ' +
 describeError(policyResult.reason) };

    if (vendorResult.status === 'rejected') {
        // Fail open: TypeError, TimeoutError, InternalError, a bad status or an
 unreadable verdict.
        headers[CHECK_HEADER] = { value: 'unavailable' };
        headers[ERROR_HEADER] = { value: describeError(vendorResult.reason) };
        return request;
    }

    const check = vendorResult.value;
    headers[CHECK_HEADER] = { value: 'ok' };
    if (policy === 'off') {
        return request;
    }

    headers[VERDICT_HEADER] = { value: check.verdict };
    headers[SCORE_HEADER] = { value: String(check.score) };
```

```
    if (check.verdict === 'bot' && policy === 'block') {
        return blockPage(check);
    }
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Fail over to a secondary origin after a health check network call in a CloudFront Functions viewer request event

The following code example shows how to probe the primary origin with cf.fetch and select a secondary origin when it is unhealthy.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// Failover for POST, which origin groups cannot do: they fail over only GET,
 HEAD and
// OPTIONS, so a POST to a primary that is down simply fails. This function
 decides before
// the request leaves the edge, with one network call: GET /health on the
 primary.
// "Slow" means the runtime's fixed 1 s TimeoutError.
//   2xx                                                -> x-route: primary
//   non-2xx, TimeoutError, InternalError, or a
//   TypeError "... could not be completed due to ..."
```

```
//   (the edge could not reach the primary)          -> x-route: fallback
//       POST, PUT, PATCH, DELETE -> the secondary origin
//       GET, HEAD                -> the static mirror
//   any other TypeError (the call itself is invalid) -> x-route: primary (fail
 open)
// GET and HEAD on the primary route also get an origin group (primary, then the
 mirror
// on 5xx). The probe costs a call per request, so it runs only when the
 KeyValueStore
// key health-probe is "on".

// Origin ids (the origins' Names in the console), all on this distribution.
const PRIMARY_ORIGIN_ID = 'primary';
const SECONDARY_ORIGIN_ID = 'secondary';
const MIRROR_ORIGIN_ID = 'static-mirror';
const HEALTH_PATH = '/health';
const PROBE_SWITCH_KEY = 'health-probe';
const FAILOVER_STATUS_CODES = [500, 502, 503, 504];
const ROUTE_HEADER = 'x-route';
const ROUTE_ERROR_HEADER = 'x-route-error';
const PROBE_HEADER = 'x-health-probe';
const PROBE_ERROR_HEADER = 'x-health-probe-error';

function describeError(error) {
    return error.name + ': ' + error.message;
}

// The runtime also uses TypeError when the edge could not reach the origin,
 naming the
// result type: "The request could not be completed due to OriginDnsError".
// That says the primary is down, so it fails over; any other TypeError is a bad
 call.
function originUnreachable(error) {
    return error.name === 'TypeError' && error.message.indexOf('could not be
 completed due to') !== -1;
}

// Never trust routing headers coming from the viewer.
function stripRoutingHeaders(headers) {
    delete headers[ROUTE_HEADER];
    delete headers[ROUTE_ERROR_HEADER];
    delete headers[PROBE_HEADER];
    delete headers[PROBE_ERROR_HEADER];
}
```

```
// The KVS key health-probe switches the probe on. No KVS, no key, or any other
 value: off.
async function probeEnabled() {
    try {
        const probeSwitch = await cf.kvs().get(PROBE_SWITCH_KEY);
        return probeSwitch === 'on';
    } catch (error) {
        return false;
    }
}

// The one network call. Resolves to { route, probe }; never throws.
async function probePrimary(headers) {
    try {
        // No body read (includeBody defaults to false): the status is the
 verdict.
        const healthResponse = await cf.fetch(HEALTH_PATH, { originId:
 PRIMARY_ORIGIN_ID });
        if (!healthResponse.ok) {
            return { route: 'fallback', probe: 'unhealthy:' +
 healthResponse.status };
        }
        return { route: 'primary', probe: 'ok' };
    } catch (error) {
        headers[PROBE_ERROR_HEADER] = { value: describeError(error) };
        // A bad call says nothing about the origin: fail open.
        if (error.name === 'TypeError' && !originUnreachable(error)) {
            return { route: 'primary', probe: 'unavailable' };
        }
        return { route: 'fallback', probe: error.name === 'TimeoutError' ?
 'unhealthy:timeout' : 'unhealthy:error' };
    }
}

// Fallback: the mirror can serve a GET or HEAD, only the secondary can take a
 write.
// Primary: a GET or HEAD still fails over to the mirror on 5xx.
function selectOrigins(route, replayable) {
    if (route === 'fallback') {
        cf.selectRequestOriginById(replayable ? MIRROR_ORIGIN_ID :
 SECONDARY_ORIGIN_ID);
    } else if (replayable) {
        cf.createRequestOriginGroup({
```

```
            originIds: [{ originId: PRIMARY_ORIGIN_ID }, { originId:
 MIRROR_ORIGIN_ID }],
            failoverCriteria: { statusCodes: FAILOVER_STATUS_CODES },
        });
    }
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    stripRoutingHeaders(headers);
    // The static mirror needs a file name.
    if (request.uri.slice(-1) === '/') {
        request.uri += 'index.html';
    }
    const replayable = request.method === 'GET' || request.method === 'HEAD';

    let health = { route: 'primary', probe: 'off' };
    const enabled = await probeEnabled();
    if (enabled) {
        health = await probePrimary(headers);
    }

    try {
        selectOrigins(health.route, replayable);
    } catch (error) {
        // The request still goes to the cache behavior's origin, the primary.
        headers[ROUTE_ERROR_HEADER] = { value: describeError(error) };
    }
    headers[ROUTE_HEADER] = { value: health.route };
    headers[PROBE_HEADER] = { value: health.probe };
    return request;
}

// Viewer response function. No network call.
// CloudFront does not run viewer response functions when the origin answered 400
 or
// higher, so this only sees successes and only annotates them:
//   x-served-by: primary    the primary set x-origin: primary
//   x-served-by: secondary  the secondary set x-origin: secondary
//   x-served-by: fallback   no marker: the static mirror answered
//   x-fallback: true        on anything but primary
// It also copies the request function's verdict headers onto the response.
```

```
// Compare x-route with x-served-by to tell "the probe decided" from "the origin
 group failed over".

const ORIGIN_MARKER_HEADER = 'x-origin';
const SERVED_BY_HEADER = 'x-served-by';
const FALLBACK_HEADER = 'x-fallback';
const PRIMARY_MARKER = 'primary';
const SECONDARY_MARKER = 'secondary';
const VERDICT_HEADERS = ['x-route', 'x-health-probe', 'x-health-probe-error', 'x-
route-error'];

function servedBy(marker) {
    if (marker === PRIMARY_MARKER) {
        return 'primary';
    }
    if (marker === SECONDARY_MARKER) {
        return 'secondary';
    }
    return 'fallback';
}

function copyVerdictHeaders(requestHeaders, responseHeaders) {
    for (let i = 0; i < VERDICT_HEADERS.length; i++) {
        const header = requestHeaders[VERDICT_HEADERS[i]];
        if (header) {
            responseHeaders[VERDICT_HEADERS[i]] = { value: header.value };
        }
    }
}

function handler(event) {
    const headers = event.response.headers;

    // The marker is internal: read it, then drop it.
    const marker = headers[ORIGIN_MARKER_HEADER] ?
 headers[ORIGIN_MARKER_HEADER].value : '';
    delete headers[ORIGIN_MARKER_HEADER];
    const origin = servedBy(marker);
    headers[SERVED_BY_HEADER] = { value: origin };
    if (origin !== 'primary') {
        headers[FALLBACK_HEADER] = { value: 'true' };
    }

    copyVerdictHeaders(event.request.headers, headers);
```

```
    return event.response;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Read configuration through KeyValueStore with a network call in a CloudFront Functions viewer request event

The following code example shows how to read a key's configuration from KeyValueStore and fetch it with cf.fetch only when the entry is missing or expired.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// Read-through config per key (a tenant, an API key, a model route). The key is
 the first
// path segment: /acme/orders reads the config of acme.
//   fresh KVS entry -> use it, no network call
//   miss or expired -> one call: GET /config/<key> on the config origin
//   call fails      -> the expired KVS entry if there is one, else
 DEFAULT_CONFIG (fail open)
// The origin counts reads per key, and a job off the request path writes hot
 keys to KVS,
// so their next reads are KVS hits. The result is forwarded in the x-config-*
 headers.
```

```
// The origin id (the origin's Name in the console) of the config service on this
 distribution.
const CONFIG_ORIGIN_ID = 'config';
const CONFIG_PATH = '/config/';
const KVS_KEY_PREFIX = 'cfg-'; // the promoter writes {"exp": <epoch seconds>,
 "config": <object>} under cfg-<key>
const DEFAULT_KEY = 'default';
const KEY_PATTERN = /^[a-z0-9-]{1,32}$/;
const DEFAULT_CONFIG = { plan: 'free', model: 'small', rateLimit: 10 };
const CONFIG_HEADER = 'x-config';
const KEY_HEADER = 'x-config-key';
const SOURCE_HEADER = 'x-config-source';
const HITS_HEADER = 'x-config-hits';
const ERROR_HEADER = 'x-config-error';
const OWN_HEADERS = [CONFIG_HEADER, KEY_HEADER, SOURCE_HEADER, HITS_HEADER,
 ERROR_HEADER];

function describeError(error) {
    return error.name + ': ' + error.message;
}

// Never trust config headers coming from the viewer: the origin trusts only what
 this function set.
function stripConfigHeaders(headers) {
    for (let i = 0; i < OWN_HEADERS.length; i++) {
        delete headers[OWN_HEADERS[i]];
    }
}

function badKey() {
    return {
        statusCode: 400,
        statusDescription: 'Bad Request',
        headers: { 'cache-control': { value: 'no-store' } },
        body: 'key must be 1-32 characters of a-z, 0-9 and -',
    };
}

// A missing key, no KVS associated, or a value that is not JSON throws: that is
 a miss.
async function readCachedEntry(key) {
    try {
        const entry = await cf.kvs().get(KVS_KEY_PREFIX + key, { format:
 'json' });
```

```
        return entry && entry.config ? entry : null;
    } catch (error) {
        return null;
    }
}

// exp is in epoch seconds, as the promoter writes it.
function isFresh(entry) {
    return entry.exp * 1000 > Date.now();
}

// The one network call, which also counts the read. Resolves to { config, hits,
 threshold }, or throws.
async function fetchConfig(key) {
    const configResponse = await cf.fetch(CONFIG_PATH + key, {
        originId: CONFIG_ORIGIN_ID,
        includeBody: true, // default false; the config is in the body
    });
    if (!configResponse.ok) {
        throw new Error('origin returned ' + configResponse.status);
    }
    const text = await configResponse.text(); // assign first: no await inside
 call arguments
    const answer = JSON.parse(text);
    if (!answer.config) {
        throw new Error('no config in the origin response');
    }
    return answer;
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    stripConfigHeaders(headers);

    const key = request.uri.split('/')[1] || DEFAULT_KEY;
    if (!KEY_PATTERN.test(key)) {
        return badKey();
    }

    // KVS first; only a missing or expired entry makes the call.
    const cached = await readCachedEntry(key);
    let config = DEFAULT_CONFIG;
    let source = 'default';
```

```
    if (cached && isFresh(cached)) {
        config = cached.config;
        source = 'kvs';
    } else {
        try {
            const fetched = await fetchConfig(key);
            config = fetched.config;
            source = 'origin';
            if (typeof fetched.hits === 'number') {
                headers[HITS_HEADER] = { value: fetched.hits + '/' +
 fetched.threshold };
            }
        } catch (error) {
            // Fail open: TypeError, TimeoutError, InternalError, a bad status or
 body. Stale beats default.
            headers[ERROR_HEADER] = { value: describeError(error) };
            if (cached) {
                config = cached.config;
                source = 'kvs-stale';
            }
        }
    }

    headers[KEY_HEADER] = { value: key };
    headers[SOURCE_HEADER] = { value: source };
    headers[CONFIG_HEADER] = { value: JSON.stringify(config) };
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Make a network call to an origin from a CloudFront Functions viewer request event

The following code example shows how to make one network call with cf.fetch to an origin on the same distribution and copy part of the answer into a request header.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// Makes one network call: GET /hello on an origin of this distribution, then
 copies one
// field of the JSON answer into a request header. If the call fails, the request
 still
// goes through, with the error in a header (fail open).

// The origin id (the origin's Name in the console) of an origin on this
 distribution.
const HELLO_ORIGIN_ID = 'hello-origin';
const HELLO_PATH = '/hello';
const MESSAGE_HEADER = 'x-hello-message';
const STATUS_HEADER = 'x-hello-status';
const ERROR_HEADER = 'x-hello-error';

async function handler(event) {
    const request = event.request;

    // Never pass on viewer-supplied copies of the headers this function sets.
    delete request.headers[MESSAGE_HEADER];
    delete request.headers[STATUS_HEADER];
    delete request.headers[ERROR_HEADER];

    try {
        const helloResponse = await cf.fetch(HELLO_PATH, {
            originId: HELLO_ORIGIN_ID,
            includeBody: true, // default false; needed to read the body
        });
        const text = await helloResponse.text(); // assign first: no await inside
 call arguments
```

```
        const hello = JSON.parse(text);
        request.headers[MESSAGE_HEADER] = { value: String(hello.message) };
        request.headers[STATUS_HEADER] = { value: String(helloResponse.status) };
    } catch (error) {
        // TypeError, TimeoutError, InternalError, or a body that is not JSON.
        request.headers[ERROR_HEADER] = { value: error.name + ': ' +
 error.message };
    }
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Serve markdown to AI agents and queue conversions with a network call from CloudFront Functions

The following code example shows how to serve AI agents a markdown copy of a page found in KeyValueStore, and queue a conversion with cf.beacon when there is none yet.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// AI agents get a markdown copy of a page, made the first time one asks for it:
//   not an agent, or not a page -> x-markdown: skipped, no call
//   KVS has md:<uri>            -> URI rewritten to the .md object, x-markdown:
 served, no call
```

```
//   no key yet                  -> the HTML as usual, plus one cf.beacon { uri,
 minute } to a
//                                  queue, x-markdown: queued (failed if
 cf.beacon threw)
// A worker behind the queue converts the page, then writes the .md and the KVS
 key.
// KVS answers "does markdown exist yet" without spending the one network call.

// The origin id (the origin's Name in the console) of the queue API on this
 distribution.
const QUEUE_ORIGIN_ID = 'markdown-queue';
const CONVERT_PATH = '/convert';
const INDEX_DOCUMENT = 'index.html';
const MARKDOWN_KEY_PREFIX = 'md:'; // md:<page uri> -> the .md object's URI,
 written by the worker
const MARKDOWN_TYPE = 'text/markdown';
const MS_PER_MINUTE = 60000;
// .html pages only, no dots elsewhere: the worker must accept exactly the same
 URIs.
const PAGE = /^\/[\w\/-]{1,200}\.html$/;
// An agent says so with Accept, or AWS WAF Bot Control labels it an AI bot. A
 web ACL rule
// writes this header on every request (the bot's name, or none), overwriting any
 viewer copy.
const WAF_AGENT_HEADER = 'x-amzn-waf-ai-agent';
const NOT_AN_AGENT = 'none';
const MARKDOWN_HEADER = 'x-markdown';
const ERROR_HEADER = 'x-markdown-error';

function headerValue(headers, name) {
    return headers[name] ? headers[name].value : '';
}

function describeError(error) {
    return error.name + ': ' + error.message;
}

function isAgent(headers) {
    const wafAgent = headerValue(headers, WAF_AGENT_HEADER);
    const asksForMarkdown = headerValue(headers,
 'accept').indexOf(MARKDOWN_TYPE) !== -1;
    return asksForMarkdown || (wafAgent !== '' && wafAgent !== NOT_AN_AGENT);
}
```

```
// The one network call, fire-and-forget. Throws only for an invalid call
 (TypeError).
function sendToQueue(message) {
    cf.beacon(CONVERT_PATH, {
        originId: QUEUE_ORIGIN_ID,
        method: 'POST',
        headers: { 'content-type': 'application/json' }, // plain strings, not
 { value } objects
        body: JSON.stringify(message),
        logDeliveryErrors: true, // a delivery failure after the function
 returned goes to CloudWatch Logs
    });
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    // Only this function tells the response side what happened: drop viewer-
supplied copies.
    delete headers[MARKDOWN_HEADER];
    delete headers[ERROR_HEADER];

    // An Amazon S3 origin has no index document below the root: a folder means
 its index.html.
    if (request.uri.slice(-1) === '/') {
        request.uri += INDEX_DOCUMENT;
    }
    const uri = request.uri;
    if (!isAgent(headers) || !PAGE.test(uri)) {
        headers[MARKDOWN_HEADER] = { value: 'skipped' };
        return request;
    }

    // 1. KVS. A missing key, or no KVS associated, throws: not converted yet.
    try {
        request.uri = await cf.kvs().get(MARKDOWN_KEY_PREFIX + uri);
        headers[MARKDOWN_HEADER] = { value: 'served' };
        return request;
    } catch (error) {
        // KeyNotFound: fall through and ask for a conversion.
    }

    // 2. Miss: the HTML goes out as usual; one message asks for markdown.
    try {
```

```
        // The URI and the minute: a FIFO queue with content-based deduplication
 turns a burst
        // into one message, and a page whose key was deleted can be queued again
 a minute later.
        sendToQueue({ uri: uri, minute: Math.floor(Date.now() /
 MS_PER_MINUTE) });
        headers[MARKDOWN_HEADER] = { value: 'queued' };
    } catch (error) {
        headers[MARKDOWN_HEADER] = { value: 'failed' };
        headers[ERROR_HEADER] = { value: describeError(error) };
    }
    return request;
}

// Viewer response function. No network call.
// An Amazon S3 origin echoes nothing, so this copies the x-markdown verdict that
 index.js left
// on the request onto the response, and adds Vary: the same URL answers HTML or
 markdown.
// CloudFront does not run viewer response functions when the origin answered 400
 or higher.

const VERDICT_HEADERS = ['x-markdown', 'x-markdown-error'];
const VARY = 'Accept, User-Agent'; // Accept, and the user agent Bot Control
 labels by

function handler(event) {
    const requestHeaders = event.request.headers;
    const response = event.response;
    for (let i = 0; i < VERDICT_HEADERS.length; i++) {
        const header = requestHeaders[VERDICT_HEADERS[i]];
        if (header) {
            response.headers[VERDICT_HEADERS[i]] = { value: header.value };
        }
    }
    response.headers['vary'] = { value: VARY };
    return response;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Normalize query string parameters in a CloudFront Functions viewer request

The following code example shows how to normalize query string parameters in a CloudFront Functions viewer request.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
function handler(event) {
     var qs=[];
     for (var key in event.request.querystring) {
         if (event.request.querystring[key].multiValue) {
             event.request.querystring[key].multiValue.forEach((mv) =>
 {qs.push(key + "=" + mv.value)});
         } else {
             qs.push(key + "=" + event.request.querystring[key].value);
         }
     };

     event.request.querystring = qs.sort().join('&');


     return event.request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Redirect to a new URL in a CloudFront Functions viewer request event

The following code example shows how to redirect to a new URL in a CloudFront Functions viewer request event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
async function handler(event) {
    var request = event.request;
    var headers = request.headers;
    var host = request.headers.host.value;
    var country = 'DE' // Choose a country code
    var newurl = `https://${host}/de/index.html`; // Change the redirect URL to
 your choice

    if (headers['cloudfront-viewer-country']) {
        var countryCode = headers['cloudfront-viewer-country'].value;
        if (countryCode === country) {
            var response = {
                statusCode: 302,
                statusDescription: 'Found',
                headers:
                    { "location": { "value": newurl } }
                }

            return response;
        }
    }
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Rewrite a request URI based on KeyValueStore configuration for a CloudFront Functions viewer request event

The following code example shows how to rewrite a request URI based on KeyValueStore configuration for a CloudFront Functions viewer request event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// (Optional) Replace KVS_ID with actual KVS ID
const kvsId = "KVS_ID";
// enable stickiness by setting a cookie from origin or using another edge
 function
const stickinessCookieName = "appversion";
// set to true to enable console logging
const loggingEnabled = false;

// function rewrites the request uri based on configuration in KVS
// example config in KVS in key:value format
// "latest": {"a_weightage": .8, "a_url": "v1", "b_url": "v2"}
// given above key and value in KVS the request uri will be rewritten
// for example http(s)://domain/latest/something/else will be rewritten as
 http(s)://domain/v1/something/else or http(s)://domain/v2/something/else
 depending on weightage
// if no configuration is found, then the request is returned as is
```

```
async function handler(event) {
    // NOTE: This example function is for a viewer request event trigger.
    // Choose viewer request for event trigger when you associate this function
 with a distribution.
    const request = event.request;
    const pathSegments = request.uri.split('/');
    const key = pathSegments[1];

    // if empty path segment or if there is valid stickiness cookie
    // then skip call to KVS and let the request continue.
    if (!key || hasValidSticknessCookie(request.cookies[stickinessCookieName],
 key)) {
        return event.request;
    }

    try {
        // get the prefix replacement from KVS
        const replacement = await getPathPrefixByWeightage(key);
        if (!replacement) {
            return event.request;
        }
        //Replace the first path with the replacement
        pathSegments[1] = replacement;
        log(`using prefix ${pathSegments[1]}`)
        const newUri = pathSegments.join('/');
        log(`${request.uri} -> ${newUri}`);
        request.uri = newUri;

        return request;
    } catch (err) {
        // No change to the path if the key is not found or any other error
        log(`request uri: ${request.uri}, error: ${err}`);
    }
    // no change to path - return request
    return event.request;
}

// function to get the prefix from KVS
async function getPathPrefixByWeightage(key) {
    const kvsHandle = cf.kvs(kvsId);
    // get the weightage config from KVS
    const kvsResponse = await kvsHandle.get(key);
    const weightageConfig = JSON.parse(kvsResponse);
    // no configuration - return null
```

```
    if (!weightageConfig || !isFinite(weightageConfig.a_weightage)) {
        return null;
    }
    // return the url based on weightage
    // return null if no url is configured
    if (Math.random() <= weightageConfig.a_weightage) {
        return weightageConfig.a_url ? weightageConfig.a_url: null;
    } else {
        return weightageConfig.b_url ? weightageConfig.b_url : null;
    }
}

// function to check if the stickiness cookie is valid
function hasValidSticknessCookie(stickinessCookie, pathSegment) {
    // if the value exists and it matches pathSegment
    return (stickinessCookie && stickinessCookie.value === pathSegment)
}

function log(message) {
    if (loggingEnabled) {
        console.log(message);
    }
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Route a request through a private service behind a VPC origin with a network call in a CloudFront Functions viewer request event

The following code example shows how to ask a private routing service behind a CloudFront VPC origin with cf.fetch which backend a tenant belongs to, and rewrite the URI.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
// Asks a private routing service, an internal Application Load Balancer reached
 through a
// CloudFront VPC origin, which backend a tenant belongs to, then rewrites the
 URI.
//   1. KVS kill switch: routing-kill-switch = off forwards unchanged, with no
 call.
//   2. One network call: GET /route?tenant=<tenant> on the VPC origin.
//   3. Rewrite the URI to /<backend><uri> and set x-backend.
// Any failure forwards unchanged, with x-routing-error (fail open).

// The origin id (the origin's Name in the console) of the VPC origin on this
 distribution.
const ROUTER_ORIGIN_ID = 'private-router';
const ROUTE_PATH = '/route?tenant=';
const KILL_SWITCH_KEY = 'routing-kill-switch';
const BACKENDS = ['blue', 'green'];
const DEFAULT_TENANT = 'guest';
const TENANT_HEADER = 'x-tenant';
const BACKEND_HEADER = 'x-backend';
const KVS_ERROR_HEADER = 'x-routing-kvs-error';
const ERROR_HEADER = 'x-routing-error';

function describeError(error) {
    return error.name + ': ' + error.message;
}

// Never trust routing headers coming from the viewer. x-tenant is read before
 this, as input.
function stripRoutingHeaders(headers) {
```

```
    delete headers[BACKEND_HEADER];
    delete headers[KVS_ERROR_HEADER];
    delete headers[ERROR_HEADER];
}

// The tenant names a route, not an identity: an x-tenant header, else "guest".
function readTenant(headers) {
    const fromHeader = headers[TENANT_HEADER];
    if (fromHeader && fromHeader.value) {
        return fromHeader.value;
    }
    return DEFAULT_TENANT;
}

// Read before the call, never in parallel: once the call has started, the switch
 cannot stop it.
// A missing key means routing is on. Any other KVS error also leaves routing on,
 and is reported.
async function routingSwitchedOff(headers) {
    try {
        const killSwitch = await cf.kvs().get(KILL_SWITCH_KEY);
        return killSwitch === 'off';
    } catch (error) {
        if (error.message.indexOf('KeyNotFound') === -1) {
            headers[KVS_ERROR_HEADER] = { value: describeError(error) };
        }
        return false;
    }
}

// The one network call. Resolves to one of BACKENDS, or throws.
async function lookupBackend(tenant) {
    const routeResponse = await cf.fetch(ROUTE_PATH + encodeURIComponent(tenant),
 {
        originId: ROUTER_ORIGIN_ID,
        includeBody: true, // default false; the backend is in the body
    });
    if (!routeResponse.ok) {
        throw new Error('route service returned ' + routeResponse.status);
    }
    const text = await routeResponse.text(); // assign first: no await inside
 call arguments
    const route = JSON.parse(text);
    if (BACKENDS.indexOf(route.backend) === -1) {
```

```
        throw new Error('unknown backend ' + route.backend);
    }
    return route.backend;
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    const tenant = readTenant(headers);
    stripRoutingHeaders(headers);
    headers[TENANT_HEADER] = { value: tenant };

    const switchedOff = await routingSwitchedOff(headers);
    if (switchedOff) {
        headers[BACKEND_HEADER] = { value: 'off' };
        return request;
    }

    try {
        const backend = await lookupBackend(tenant);
        request.uri = '/' + backend + request.uri;
        headers[BACKEND_HEADER] = { value: backend };
    } catch (error) {
        // Fail open: TypeError, TimeoutError, InternalError, or a bad answer.
        headers[ERROR_HEADER] = { value: describeError(error) };
    }
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Route requests to an origin closer to the viewer in a CloudFront Functions viewer request event

The following code example shows how to route requests to an origin closer to the viewer in a CloudFront Functions viewer request event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

function handler(event) {
    const request = event.request;
    const headers = request.headers;
    const country = headers['cloudfront-viewer-country'] &&
        headers['cloudfront-viewer-country'].value;

    //List of Regions with S3 buckets containing content
    const countryToRegion = {
        'DE': 'eu-central-1',
        'IE': 'eu-west-1',
        'GB': 'eu-west-2',
        'FR': 'eu-west-3',
        'JP': 'ap-northeast-1',
        'IN': 'ap-south-1'
    };

    const DEFAULT_REGION = 'us-east-1';

    const selectedRegion = (country && countryToRegion[country]) ||
 DEFAULT_REGION;

    const domainName =
        `cloudfront-functions-demo-bucket-in-${selectedRegion}.s3.
${selectedRegion}.amazonaws.com`;

    cf.updateRequestOrigin({
        "domainName": domainName,
        "originAccessControlConfig": {
            "enabled": true,
            "region": selectedRegion,
```

```
            "signingBehavior": "always",
            "signingProtocol": "sigv4",
            "originType": "s3"
        },
    });

    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Send fire-and-forget events with a network call from CloudFront Functions

The following code example shows how to send a fire-and-forget event with cf.beacon from viewer request and viewer response events.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';
import crypto from 'crypto';

// Viewer request function.
// Fires one small JSON event at an origin of this distribution with cf.beacon
// (fire-and-forget), then forwards the request whatever happened.
//   x-signal: beacon -> handed off; nothing to await, no answer to read
//   x-signal: failed -> cf.beacon threw (x-signal-error)
// cf.beacon counts as the one network call of this invocation.
```

```
// The origin id (the origin's Name in the console) of the event collector on
 this distribution.
const SIGNAL_ORIGIN_ID = 'signal-collector';
const SIGNAL_PATH = '/signal';
const MAX_SIGNAL_ID_LENGTH = 128; // fits an SQS FIFO deduplication id
const MAX_USER_AGENT_LENGTH = 200;
const VIEWER_HASH_LENGTH = 16;
const SIGNAL_ID_HEADER = 'x-signal-id';
const SIGNAL_HEADER = 'x-signal';
const ERROR_HEADER = 'x-signal-error';

// Never send the raw viewer IP: a truncated hash is enough to tell viewers
 apart.
function hashViewerIp(ip) {
    return crypto.createHash('sha256').update(ip).digest('hex').slice(0,
 VIEWER_HASH_LENGTH);
}

function requestedSignal(event, signalId) {
    const headers = event.request.headers;
    const userAgent = headers['user-agent'] ? headers['user-agent'].value : '';
    return {
        kind: 'requested',
        uri: signalId,
        ip: hashViewerIp(event.viewer.ip),
        ua: userAgent.slice(0, MAX_USER_AGENT_LENGTH),
        ts: Date.now(),
    };
}

async function handler(event) {
    const request = event.request;

    // Never pass on viewer-supplied copies of the headers this function sets.
    delete request.headers[SIGNAL_ID_HEADER];
    delete request.headers[SIGNAL_HEADER];
    delete request.headers[ERROR_HEADER];

    // The URI is the deduplication id; carried forward so response-function.js
 can correlate.
    const signalId = request.uri.slice(0, MAX_SIGNAL_ID_LENGTH);
    request.headers[SIGNAL_ID_HEADER] = { value: signalId };

    try {
```

```
        cf.beacon(SIGNAL_PATH, {
            originId: SIGNAL_ORIGIN_ID,
            method: 'POST',
            headers: { 'content-type': 'application/json' }, // plain strings,
 not { value } objects
            body: JSON.stringify(requestedSignal(event, signalId)),
            logDeliveryErrors: true, // a delivery failure after the function
 returned goes to CloudWatch Logs
        });
        request.headers[SIGNAL_HEADER] = { value: 'beacon' };
    } catch (error) {
        // Only an invalid call throws here (TypeError): a bad option, or a
 second call in one invocation.
        request.headers[SIGNAL_HEADER] = { value: 'failed' };
        request.headers[ERROR_HEADER] = { value: error.name + ': ' +
 error.message };
    }
    return request;
}

import cf from 'cloudfront';
import crypto from 'crypto';

// Viewer response function.
// The viewer response event runs on every response, cache hits included, so a
 network
// call here must be rare by construction: this function fires a "served" event
 with
// cf.beacon only for a sample of requests, chosen from a hash of the request id.
// Everything else gets x-signal-served: skipped and no call.
// The sample is a hash, not Math.random(): a function cannot make a network call
 after
// it has generated random values.

// The origin id (the origin's Name in the console) of the event collector on
 this distribution.
const SIGNAL_ORIGIN_ID = 'signal-collector';
const SIGNAL_PATH = '/signal';
// Share of responses that send a "served" event, in percent.
const SAMPLE_PERCENT = 1;
const MAX_SIGNAL_ID_LENGTH = 128; // fits an SQS FIFO deduplication id
const SIGNAL_ID_HEADER = 'x-signal-id';
const SERVED_HEADER = 'x-signal-served';
const ERROR_HEADER = 'x-signal-served-error';
```

```
function headerValue(headers, name) {
    return headers[name] ? headers[name].value : '';
}

// The first 16 bits of SHA-256(request id), as a share of 65536.
function sampled(requestId) {
    if (!requestId) {
        return false;
    }
    const bucket =
 parseInt(crypto.createHash('sha256').update(requestId).digest('hex').slice(0,
 4), 16);
    return bucket < (SAMPLE_PERCENT / 100) * 65536;
}

function servedSignal(request, status) {
    const uri = request.uri.slice(0, MAX_SIGNAL_ID_LENGTH);
    return {
        kind: 'served',
        uri: uri,
        status: status,
        // Correlates with the "requested" event that index.js sent on the way
 in.
        signalId: headerValue(request.headers, SIGNAL_ID_HEADER) || uri,
        // Its own deduplication id, so it never collapses into the "requested"
 event.
        dedupId: ('served:' + uri + ':' + status).slice(0, MAX_SIGNAL_ID_LENGTH),
        ts: Date.now(),
    };
}

async function handler(event) {
    const request = event.request;
    const response = event.response;

    if (!sampled(event.context.requestId)) {
        response.headers[SERVED_HEADER] = { value: 'skipped' };
        return response;
    }

    try {
        cf.beacon(SIGNAL_PATH, {
            originId: SIGNAL_ORIGIN_ID,
```

```
            method: 'POST',
            headers: { 'content-type': 'application/json' }, // plain strings,
 not { value } objects
            body: JSON.stringify(servedSignal(request, response.statusCode)),
            logDeliveryErrors: true, // a delivery failure after the function
 returned goes to CloudWatch Logs
        });
        response.headers[SERVED_HEADER] = { value: 'beacon' };
    } catch (error) {
        response.headers[SERVED_HEADER] = { value: 'failed' };
        response.headers[ERROR_HEADER] = { value: error.name + ': ' +
 error.message };
    }
    return response;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Assign a sticky A/B variant with KeyValueStore and a network call in CloudFront Functions

The following code example shows how to read a user's A/B variant from KeyValueStore, assign one with cf.fetch only on a miss, and set the variant cookies in a viewer response event.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';
import crypto from 'crypto';
```

```
// Viewer request function.
// Sticky A/B variant per user: the uid cookie names the user, the KeyValueStore
 holds the
// warm working set, and an assignment service behind an origin is the source of
 truth.
//   KVS hit  -> x-ab-variant from KVS, no network call
//   KVS miss -> one call: POST /assign/<uid> on the assignment origin
//   failure  -> x-ab-variant: control and x-ab-error, request still forwarded
 (fail open)
// This side never sets a cookie: response-function.js turns x-ab-set-uid and
// x-ab-set-variant into Set-Cookie, so stickiness survives cache hits.

// The origin id (the origin's Name in the console) of the assignment service on
 this distribution.
const ASSIGNER_ORIGIN_ID = 'assigner';
const ASSIGN_PATH = '/assign/';
const UID_COOKIE = 'uid';
const UID_PATTERN = /^[a-f0-9]{16}$/;
const UID_LENGTH = 16;
const VARIANTS = ['A', 'B'];
const FALLBACK_VARIANT = 'control';
// Every header this function sets starts with this prefix; viewer-supplied ones
 are deleted.
const AB_HEADER_PREFIX = 'x-ab-';
// Read by response-function.js: only this function may set them.
const SET_UID_HEADER = 'x-ab-set-uid';
const SET_VARIANT_HEADER = 'x-ab-set-variant';
// Forwarded to the origin.
const UID_HEADER = 'x-ab-uid';
const NEW_UID_HEADER = 'x-ab-uid-new';
const KVS_HEADER = 'x-ab-kvs';
const VARIANT_HEADER = 'x-ab-variant';
const SOURCE_HEADER = 'x-ab-source';
const ERROR_HEADER = 'x-ab-error';

function describeError(error) {
    return error.name + ': ' + error.message;
}

function isVariant(value) {
    return VARIANTS.indexOf(value) !== -1;
}
```

```
// Never trust A/B headers coming from the viewer: the response side turns two of
 them into cookies.
function stripAbHeaders(headers) {
    for (const name in headers) {
        if (name.indexOf(AB_HEADER_PREFIX) === 0) {
            delete headers[name];
        }
    }
}

// Unique enough per browser. Not Math.random(): a function cannot make a network
 call
// after generating random values, so it hashes the request id instead.
function newUid(event) {
    const seed = event.context.requestId + event.viewer.ip + Date.now();
    return crypto.createHash('sha256').update(seed).digest('hex').slice(0,
 UID_LENGTH);
}

// The viewer's uid cookie, or a new uid that the response side will set as a
 cookie.
function readOrIssueUid(event) {
    const request = event.request;
    const cookie = request.cookies[UID_COOKIE];
    const uid = cookie ? cookie.value : '';
    if (UID_PATTERN.test(uid)) {
        return uid;
    }
    const issued = newUid(event);
    request.cookies[UID_COOKIE] = { value: issued };
    request.headers[NEW_UID_HEADER] = { value: '1' };
    request.headers[SET_UID_HEADER] = { value: issued };
    return issued;
}

// A missing key, or no KVS associated, throws: that is a miss.
async function readVariant(uid, headers) {
    try {
        const variant = await cf.kvs().get(uid);
        headers[KVS_HEADER] = { value: 'hit' };
        return variant;
    } catch (error) {
        headers[KVS_HEADER] = { value: 'miss: ' + error.message };
        return '';
```

```
    }
}

// The one network call, on a KVS miss. Resolves to A or B, or throws.
async function assignVariant(uid) {
    const assignResponse = await cf.fetch(ASSIGN_PATH + uid, {
        originId: ASSIGNER_ORIGIN_ID,
        method: 'POST',
        includeBody: true, // default false; the variant is in the body
    });
    if (!assignResponse.ok) {
        throw new Error('assigner returned ' + assignResponse.status);
    }
    const text = await assignResponse.text(); // assign first: no await inside
 call arguments
    const assignment = JSON.parse(text);
    if (!isVariant(assignment.variant)) {
        throw new Error('assigner variant unreadable');
    }
    return assignment.variant;
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    stripAbHeaders(headers);

    const uid = readOrIssueUid(event);
    headers[UID_HEADER] = { value: uid };

    // 1. KVS first: a hit needs no call.
    let variant = await readVariant(uid, headers);
    let source = 'kvs';

    // 2. Miss: one call to the assigner.
    if (!isVariant(variant)) {
        try {
            variant = await assignVariant(uid);
            source = 'origin';
        } catch (error) {
            // Fail open: TypeError, TimeoutError, InternalError, a bad status or
 body.
            variant = FALLBACK_VARIANT;
            source = 'fallback';
```

```
            headers[ERROR_HEADER] = { value: describeError(error) };
        }
    }

    headers[VARIANT_HEADER] = { value: variant };
    headers[SOURCE_HEADER] = { value: source };
    // control is not an assignment, so it never becomes a cookie.
    if (variant !== FALLBACK_VARIANT) {
        headers[SET_VARIANT_HEADER] = { value: variant };
    }
    return request;
}

// Viewer response function. No network call, no KVS read.
// The request function decided (KVS first, one cf.fetch on a miss) and left the
 decision
// on the request as x-ab-set-uid and x-ab-set-variant. This side only turns
 those into
// cookies. It runs on cache hits too, so it must stay cheap. Both values are
 checked again.

const SET_UID_HEADER = 'x-ab-set-uid';
const SET_VARIANT_HEADER = 'x-ab-set-variant';
const UID_COOKIE = 'uid';
const VARIANT_COOKIE = 'ab';
const UID_PATTERN = /^[a-f0-9]{16}$/;
const VARIANTS = ['A', 'B'];
const COOKIE_ATTRIBUTES = 'Path=/; Max-Age=86400; Secure; HttpOnly;
 SameSite=Lax';

function headerValue(headers, name) {
    return headers[name] ? headers[name].value : '';
}

function handler(event) {
    const requestHeaders = event.request.headers;
    const response = event.response;
    const uid = headerValue(requestHeaders, SET_UID_HEADER);
    const variant = headerValue(requestHeaders, SET_VARIANT_HEADER);

    if (UID_PATTERN.test(uid)) {
        response.cookies[UID_COOKIE] = { value: uid, attributes:
 COOKIE_ATTRIBUTES };
    }
```

```
    if (VARIANTS.indexOf(variant) !== -1) {
        response.cookies[VARIANT_COOKIE] = { value: variant, attributes:
 COOKIE_ATTRIBUTES };
    }
    return response;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use key-value pairs in a CloudFront Functions viewer request

The following code example shows how to use key-value pairs in a CloudFront Functions viewer request.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// This fails if there is no key value store associated with the function
const kvsHandle = cf.kvs();

// Remember to associate the KVS with your function before referencing KVS in
 your code.
// https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/kvs-with-
functions-associate.html
async function handler(event) {
    const request = event.request;
    // Use the first segment of the pathname as key
    // For example http(s)://domain/<key>/something/else
    const pathSegments = request.uri.split('/')
```

```
    const key = pathSegments[1]
    try {
        // Replace the first path of the pathname with the value of the key
        // For example http(s)://domain/<value>/something/else
        pathSegments[1] = await kvsHandle.get(key);
        const newUri = pathSegments.join('/');
        console.log(`${request.uri} -> ${newUri}`)
        request.uri = newUri;
    } catch (err) {
        // No change to the pathname if the key is not found
        console.log(`${request.uri} | ${err}`);
    }
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Validate an Amazon Cognito access token with a network call in a CloudFront Functions viewer request event

The following code example shows how to validate a bearer token with the Amazon Cognito GetUser API through cf.fetch and forward the user's identity headers, or deny the request.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';

// Viewer request function.
```

```
// Validates the viewer's Amazon Cognito access token with one network call to
 the Cognito
// GetUser API (no SigV4: the access token is the credential), then forwards the
 request
// with x-user-sub and x-user-name set. Anything else is denied before the
 origin:
//   no Bearer token          -> 401 missing_token, no call
//   token Cognito rejects    -> 401 invalid_token
//   Cognito cannot be asked  -> 503 auth_unavailable (fail closed)

// The origin id (the origin's Name in the console) of the regional Cognito
 endpoint,
// cognito-idp.<Region>.amazonaws.com, on this distribution.
const COGNITO_ORIGIN_ID = 'cognito';
const GET_USER_TARGET = 'AWSCognitoIdentityProviderService.GetUser';
const BEARER_PREFIX = 'bearer ';
// Every header this function sets starts with this prefix; viewer-supplied ones
 are deleted.
const IDENTITY_HEADER_PREFIX = 'x-user-';
const USER_SUB_HEADER = 'x-user-sub';
const USER_NAME_HEADER = 'x-user-name';
const ERROR_HEADER = 'x-cognito-auth-error';
const STATUS_TEXT = { 401: 'Unauthorized', 503: 'Service Unavailable' };

function describeError(error) {
    return error.name + ': ' + error.message;
}

// Never trust identity headers coming from the viewer.
function stripIdentityHeaders(headers) {
    for (const name in headers) {
        if (name.indexOf(IDENTITY_HEADER_PREFIX) === 0) {
            delete headers[name];
        }
    }
}

// The token after "Bearer ", or null when the viewer sent no Bearer credential
 at all.
function readBearerToken(headers) {
    const authorization = headers.authorization ? headers.authorization.value :
 '';
    if (authorization.slice(0, BEARER_PREFIX.length).toLowerCase() !==
 BEARER_PREFIX) {
```

```
        return null;
    }
    return authorization.slice(BEARER_PREFIX.length).trim();
}

// The one network call. Resolves to the Cognito user, or null for a token
 Cognito rejects;
// throws when Cognito cannot be asked.
async function getCognitoUser(token) {
    const cognitoResponse = await cf.fetch('/', {
        originId: COGNITO_ORIGIN_ID,
        method: 'POST',
        headers: {
            'content-type': 'application/x-amz-json-1.1',
            'x-amz-target': GET_USER_TARGET,
        }, // plain strings, not { value } objects
        body: JSON.stringify({ AccessToken: token }),
        includeBody: true, // default false; the user is in the body
    });
    // Cognito answers 400 NotAuthorizedException for a bad or expired token.
    if (!cognitoResponse.ok) {
        return null;
    }
    const text = await cognitoResponse.text(); // assign first: no await inside
 call arguments
    return JSON.parse(text);
}

// The value of a Cognito user attribute, '' when the user has none.
function userAttribute(user, name) {
    const attributes = user.UserAttributes || [];
    for (let i = 0; i < attributes.length; i++) {
        if (attributes[i].Name === name) {
            return attributes[i].Value;
        }
    }
    return '';
}

function deny(status, code, error) {
    const headers = {
        'content-type': { value: 'application/json' },
        'cache-control': { value: 'no-store' },
    };
```

```
    if (status === 401) {
        headers['www-authenticate'] = { value: 'Bearer' };
    }
    if (error) {
        headers[ERROR_HEADER] = { value: error };
    }
    return {
        statusCode: status,
        statusDescription: STATUS_TEXT[status],
        headers: headers,
        body: JSON.stringify({ error: code }),
    };
}

async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    stripIdentityHeaders(headers);

    const token = readBearerToken(headers);
    if (token === null) {
        return deny(401, 'missing_token');
    }

    let user;
    try {
        user = await getCognitoUser(token);
    } catch (error) {
        // Fail closed: TypeError, TimeoutError, InternalError, or a body that is
 not JSON.
        return deny(503, 'auth_unavailable', describeError(error));
    }

    const sub = user ? userAttribute(user, 'sub') : '';
    if (!sub) {
        return deny(401, 'invalid_token');
    }

    headers[USER_SUB_HEADER] = { value: sub };
    headers[USER_NAME_HEADER] = { value: String(user.Username) };
    return request;
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Validate a simple token in a CloudFront Functions viewer request

The following code example shows how to validate a simple token in a CloudFront Functions viewer request.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';
import crypto from 'crypto';

// Response when JWT is not valid
const response401 = { statusCode: 401, statusDescription: 'Unauthorized' };

// Remember to associate the KVS with your function before calling the const
 kvsKey = 'jwt.secret'.
// https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/kvs-with-
functions-associate.html
const kvsKey = 'jwt.secret';
// set to true to enable console logging
const loggingEnabled = false;

function jwtVerify(token, key) {
    // check token
    if (!token) {
        throw new Error('No token supplied');
    }
    // check segments
    const segments = token.split('.');
    if (segments.length !== 3) {
        throw new Error('Not enough or too many segments');
```

```
    }

    // Verify the HS256 signature
    const expected = crypto.createHmac('sha256', key).update(`${segments[0]}.
${segments[1]}`).digest('base64url');
    if (!_constantTimeEquals(segments[2], expected)) {
        throw new Error('Signature verification failed');
    }

    // base64 decode and parse JSON
    const payload = JSON.parse(Buffer.from(segments[1], 'base64url'));

    // Support for nbf and exp claims.
    // According to the RFC, they should be in seconds.
    if (payload.nbf && Date.now() < payload.nbf*1000) {
        throw new Error('Token not yet active');
    }

    if (!payload.exp || Date.now() > payload.exp*1000) {
        throw new Error('Token expired or missing exp');
    }

    return payload;
}

// Constant-time comparison to prevent timing side channels
function _constantTimeEquals(a, b) {
    if (a.length !== b.length) {
        return false;
    }
    let xor = 0;
    for (let i = 0; i < a.length; i++) {
        xor |= a.charCodeAt(i) ^ b.charCodeAt(i);
    }
    return xor === 0;
}

async function handler(event) {
    const request = event.request;

    // Secret key used to verify the JWT, read from KeyValueStore
    const secretKey = await getSecret();

    if (!secretKey) {
```

```
        return response401;
    }

    // No JWT in the query string, return 401
    if (!request.querystring.jwt) {
        log("Error: No JWT in the querystring");
        return response401;
    }

    const jwtToken = request.querystring.jwt.value;

    try {
        jwtVerify(jwtToken, secretKey);
    } catch (e) {
        log(e.message);
        return response401;
    }

    // Remove the JWT from the query string if valid and return
    delete request.querystring.jwt;
    log("Valid JWT token");
    return request;
}

// get secret from key value store
async function getSecret() {
    // initialize cloudfront kv store and get the key value
    try {
        const kvsHandle = cf.kvs();
        return await kvsHandle.get(kvsKey);
    } catch (err) {
        log(`Error reading value for key: ${kvsKey}, error: ${err}`);
        return null;
    }
}

function log(message) {
    if (loggingEnabled) {
        console.log(message);
    }
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Run a virtual waiting room with a network call in a CloudFront Functions viewer request event

The following code example shows how to draw a waiting room ticket with one cf.fetch call per viewer, and admit viewers by comparing a signed cookie with a watermark in KeyValueStore.

JavaScript

JavaScript runtime 2.0 for CloudFront Functions

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the CloudFront Functions examples repository.

```
import cf from 'cloudfront';
import crypto from 'crypto';

// Viewer request function.
// A virtual waiting room. The queue is two integers: a ticket counter behind an
 origin and a
// watermark (serving) in KeyValueStore.
//   valid pass cookie (wr_a)          -> forwarded, x-wr: admitted, no call
//   ticket cookie (wr_t) >= serving   -> waiting page with the position, no call
//   ticket cookie (wr_t) <  serving   -> 302 to the same URL with a new pass, no
 call
//   no ticket, browser page load      -> one call: POST /enqueue draws a ticket
 (fail closed)
//   no ticket, anything else          -> 503 browsers-only, no call
//   GET <prefix>leave                 -> cookies cleared, one cf.beacon frees
 the place
// An admission job raises the watermark in KVS; AWS WAF guards the counter with
 a CAPTCHA.

// The origin id (the origin's Name in the console) of the queue API on this
 distribution.
```

```
const QUEUE_ORIGIN_ID = 'waiting-room';
const ENQUEUE_PATH = '/enqueue';
const LEAVE_PATH = '/leave';
// The path the waiting room protects: cookies are scoped to it, and
 <prefix>leave leaves the line.
const PATH_PREFIX = '/';
const STATE_KEY = 'queue-state'; // from the admission job: { mode, epoch,
 serving, step, tick }
const SECRET_KEYS = ['queue-secret-cur', 'queue-secret-prev']; // cur signs,
 either verifies
const SHARDS = 8; // counter items; the admission job reads the same number
const ADMIT_SECONDS = 120; // wr_a lifetime; renewing it is free while wr_t is
 below the watermark
const TICKET_SECONDS = 7200;
const STALE_MS = 5 * 60 * 1000; // queue-state older than this: the admission job
 stopped ticking
const REFRESH_SECONDS = 5;
const TICKET_COOKIE = 'wr_t';
const PASS_COOKIE = 'wr_a';
const WAF_TOKEN_COOKIE = 'aws-waf-token';
const COOKIE_ATTRIBUTES = 'Path=' + PATH_PREFIX + '; Secure; HttpOnly;
 SameSite=Lax';
const JWT_HEADER = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.'; //
 {"alg":"HS256","typ":"JWT"}, the only header accepted
// Inserted by an AWS WAF count rule from the Bot Control token id label; WAF
 overwrites any viewer copy.
const WAF_TOKEN_ID_HEADER = 'x-amzn-waf-wr-token-id';
const JA4_HEADER = 'cloudfront-viewer-ja4-fingerprint';
const DOC = '<!doctype html><meta charset="utf-8">';
const VERDICT_HEADER = 'x-wr';
const PLACE_HEADER = 'x-wr-place';

function headerValue(headers, name) {
    return headers[name] ? headers[name].value : '';
}

function sign(secret, text) {
    return crypto.createHmac('sha256', secret).update(text).digest('base64url');
}

function sha256(text, encoding) {
    return crypto.createHash('sha256').update(text).digest(encoding);
}
```

```
function signJwt(claims, secret) {
    const signed = JWT_HEADER +
 Buffer.from(JSON.stringify(claims)).toString('base64url');
    return signed + '.' + sign(secret, signed);
}

// Constant time: timing never tells how much of a signature matched.
function constantTimeEquals(expected, actual) {
    let difference = expected.length ^ actual.length;
    for (let i = 0; i < expected.length; i++) {
        difference |= expected.charCodeAt(i) ^ actual.charCodeAt(i);
    }
    return difference === 0;
}

// The claims of a token this queue signed, or null.
function verifyJwt(cookie, secrets) {
    const token = cookie ? cookie.value : '';
    const dot = token.lastIndexOf('.');
    if (token.indexOf(JWT_HEADER) !== 0 || dot <= JWT_HEADER.length) {
        return null;
    }
    let valid = false;
    for (let i = 0; i < secrets.length; i++) {
        if (constantTimeEquals(sign(secrets[i], token.slice(0, dot)),
 token.slice(dot + 1))) {
            valid = true;
        }
    }
    if (!valid) {
        return null;
    }
    try {
        return JSON.parse(Buffer.from(token.slice(JWT_HEADER.length, dot),
 'base64url').toString());
    } catch (error) {
        return null;
    }
}

// no-store: any answer may carry a ticket or a pass.
function respond(statusCode, verdict, extraHeaders, body) {
    const headers = {
        'cache-control': { value: 'no-store' },
```

```
        'content-type': { value: 'text/html; charset=utf-8' },
    };
    headers[VERDICT_HEADER] = { value: verdict };
    const response = { statusCode: statusCode, headers: Object.assign(headers,
 extraHeaders) };
    if (body) {
        response.body = body;
    }
    return response;
}

function withCookie(response, name, value, maxAge) {
    response.cookies = response.cookies || {};
    response.cookies[name] = { value: value, attributes: COOKIE_ATTRIBUTES + ';
 Max-Age=' + maxAge };
    return response;
}

function waitingPage(verdict, title, line) {
    const html = DOC + '<meta http-equiv="refresh" content="' + REFRESH_SECONDS +
 '">' +
        '<title>' + title + '</title><body style="font:18px system-ui;max-
width:34em;margin:4em auto">' +
        '<h1>' + title + '</h1><p>' + line + '</p>' +
        '<p><a href="' + PATH_PREFIX + 'leave">Leave the line</a></p>';
    return respond(200, verdict, { 'retry-after': { value:
 String(REFRESH_SECONDS) } }, html);
}

function expiredCookie(domain) {
    return { value: '', attributes: 'Path=/; Secure; ' + domain + 'Max-Age=0' };
}

// Clears the HttpOnly cookies and aws-waf-token (both Domain forms), so the
 CAPTCHA comes back.
function leftPage(host, place) {
    const extraHeaders = {};
    extraHeaders[PLACE_HEADER] = { value: place };
    const page = respond(200, 'left', extraHeaders,
        DOC + '<title>You left the line</title><p>You left the line. <a href="' +
 PATH_PREFIX + '">Join again</a></p>');
    withCookie(withCookie(page, TICKET_COOKIE, '', 0), PASS_COOKIE, '', 0);
    const wafToken = [expiredCookie(''), expiredCookie('Domain=' + host + '; ')];
```

```
    page.cookies[WAF_TOKEN_COOKIE] = Object.assign(expiredCookie(''),
 { multiValue: wafToken });
    return page;
}

// KVS reads, not network calls; a missing key leaves state null or a secret out.
// allSettled, not all: a missing previous secret must not throw away the state.
async function readQueue() {
    const store = cf.kvs();
    const reads = await Promise.allSettled([
        store.get(STATE_KEY, { format: 'json' }),
        store.get(SECRET_KEYS[0]),
        store.get(SECRET_KEYS[1]),
    ]);
    const secrets = [];
    for (let i = 1; i < reads.length; i++) {
        if (reads[i].status === 'fulfilled' && reads[i].value) {
            secrets.push(reads[i].value);
        }
    }
    return { state: reads[0].status === 'fulfilled' ? reads[0].value : null,
 secrets: secrets };
}

// One cf.beacon puts the ticket in a skip set (a set: a replayed cookie frees
 one place).
function leaveLine(state, ticket, host) {
    if (ticket < 0 || ticket < state.serving) {
        return leftPage(host, 'none');
    }
    try {
        cf.beacon(LEAVE_PATH, {
            originId: QUEUE_ORIGIN_ID,
            method: 'POST',
            headers: { 'content-type': 'application/json' }, // plain strings,
 not { value } objects
            body: JSON.stringify({ epoch: state.epoch, ticket: ticket }),
            logDeliveryErrors: true, // late failures go to CloudWatch Logs
        });
        return leftPage(host, 'freed');
    } catch (error) {
        return leftPage(host, 'failed'); // only an invalid call throws
    }
}
```

```
// wr_a binds to a hash of the WAF token id, the user agent and JA4: a copied
 pass fails elsewhere.
function clientBinding(headers) {
    const client = [headerValue(headers, WAF_TOKEN_ID_HEADER),
 headerValue(headers, 'user-agent'),
        headerValue(headers, JA4_HEADER)].join('|');
    return sha256(client, 'base64url').slice(0, 22);
}

function positionPage(ticket, state, now) {
    if (now * 1000 - state.tick > STALE_MS) {
        return waitingPage('paused', 'The queue is paused', 'You keep your place
 in line.');
    }
    const ahead = ticket - state.serving + 1;
    const minutes = Math.ceil(ahead / Math.max(state.step, 1));
    return waitingPage('waiting', 'You are in line', 'Position about <b>' + ahead
 + '</b>, roughly ' + minutes + ' min.');
}

// The one network call: the shard's counter hands out its next number, or
 throws.
async function drawTicket(epoch, shard) {
    const enqueueResponse = await cf.fetch(ENQUEUE_PATH, {
        originId: QUEUE_ORIGIN_ID,
        method: 'POST',
        headers: { 'content-type': 'application/json' }, // plain strings, not
 { value } objects
        body: JSON.stringify({ epoch: epoch, shard: shard }),
        includeBody: true, // default false; the counter is in the body
    });
    if (!enqueueResponse.ok) {
        throw new Error('enqueue returned ' + enqueueResponse.status); // 429:
 the API throttled
    }
    const text = await enqueueResponse.text(); // assign first: no await inside
 call arguments
    const ticket = (JSON.parse(text).n - 1) * SHARDS + shard;
    if (!(ticket >= 0)) {
        throw new Error('enqueue sent no counter');
    }
    return ticket;
}
```

```
async function handler(event) {
    const request = event.request;
    const headers = request.headers;
    const now = Math.floor(Date.now() / 1000); // JWT times are in seconds
    delete headers[VERDICT_HEADER]; // only this function tells the origin what
 happened

    const queue = await readQueue();
    const state = queue.state;
    const secrets = queue.secrets;
    // The epoch makes a counter reset safe: tickets of an old line stop
 counting.
    const claims = state ? verifyJwt(request.cookies[TICKET_COOKIE], secrets) :
 null;
    const ticket = claims && claims.epoch === state.epoch && typeof claims.ticket
 === 'number' ? claims.ticket : -1;

    if (request.uri === PATH_PREFIX + 'leave') {
        return leaveLine(state, ticket, headerValue(headers, 'host'));
    }
    if (state && state.mode === 'off') {
        headers[VERDICT_HEADER] = { value: 'off' };
        return request;
    }
    if (!state || !secrets.length) {
        return waitingPage('starting', 'The queue is starting', 'Please wait a
 moment.');
    }

    const binding = clientBinding(headers);
    const pass = verifyJwt(request.cookies[PASS_COOKIE], secrets);
    if (pass && pass.epoch === state.epoch && pass.exp > now && pass.bnd ===
 binding) {
        headers[VERDICT_HEADER] = { value: 'admitted' };
        return request;
    }

    // In line. Tickets below the watermark are admitted.
    if (ticket >= 0) {
        if (ticket >= state.serving) {
            return positionPage(ticket, state, now);
        }
```

```
        const admitted = respond(302, 'admit', { location: { value:
 request.uri } });
        const newPass = signJwt({ epoch: state.epoch, exp: now + ADMIT_SECONDS,
 bnd: binding }, secrets[0]);
        return withCookie(admitted, PASS_COOKIE, newPass, ADMIT_SECONDS);
    }

    // No ticket. Only a browser page load may draw one.
    if (request.method !== 'GET' || headerValue(headers, 'accept').indexOf('text/
html') === -1) {
        return respond(503, 'browsers-only',
            { 'content-type': { value: 'application/json' }, 'retry-after':
 { value: String(REFRESH_SECONDS) } },
            '{"waitingRoom":"open this URL in a browser"}');
    }

    // A hash of the request id, not Math.random(): no network call may follow
 random values.
    const shard = parseInt(sha256(event.context.requestId, 'hex').slice(0, 8),
 16) % SHARDS;
    let drawn;
    try {
        drawn = await drawTicket(state.epoch, shard);
    } catch (error) {
        // Fail closed: no ticket, nothing reaches the origin, the page retries.
        return waitingPage('busy', 'Busy, retrying', 'The line is very long right
 now (' + error.name + ').');
    }
    const queued = waitingPage('queued', 'You are in line', 'Your ticket is <b>'
 + drawn + '</b>. Finding your position.');
    const newTicket = signJwt({ epoch: state.epoch, ticket: drawn, iat: now },
 secrets[0]);
    return withCookie(queued, TICKET_COOKIE, newTicket, TICKET_SECONDS);
}
```

For a complete list of AWS SDK developer guides and code examples, see Using CloudFront with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Document history

The following table describes the important changes made to CloudFront documentation. For notification of updates, you can subscribe to the RSS feed.

Change	Description	Date
Added viewer fallback for connection groups	You can now choose the certificate that CloudFront uses for viewers that don't send Server Name Indication (SNI) on a connection group's Anycast static IP addresses.	November 19, 2026
Added origin HTTP version	You can now have CloudFront use HTTP/2 to connect to custom origins and VPC origins that support it.	November 19, 2026
Added domains without a certificate for distribution tenants	You can now add a domain to a distribution tenant without a certificate. CloudFront verifies the domain's DNS records and serves the domain over HTTP, so that you can also validate the domain with a third-party certificate authority.	November 19, 2026
Added domain moves when you create or update a distribution tenant	You can now move domains that are associated with another standard distribution or distribution tenant to a distribution tenant when you create or update the distribution tenant, without disabling the source.	November 19, 2026

Change	Description	Date
Added connection logs for distribution tenants on Anycast static IPs	CloudFront now generates connection logs for HTTPS connections to distribut ion tenants on the Anycast static IP addresses of their connection group, in addition to mutual TLS connections.	November 19, 2026
Added address-based selection for distribution tenants	You can now serve a wildcard domain on only the Anycast static IP addresses of a distribution tenant's connection group, so that more than one distribution tenant can serve the same domain on different Anycast static IP lists.	November 19, 2026
Added HTTP/2 to custom and VPC origins	You can now have CloudFron t use HTTP/2 to connect to custom origins and VPC origins that support it.	November 19, 2026
Added access log fields for standard logging (v2)	You can now select Common Media Client Data (CMCD) fields and additional access log fields (`s-ip`, `primary-distribution-id`, `primary-distributi on-dns-name`, `r-host`, `sr-reason`, and `x-edge-mqcs`) for standard logging (v2).	November 5, 2026

Change	Description	Date
Updated quotas for multi-tenant distributions	Increased the default quotas for distribution tenants per AWS account (from 10,000 to 1,000,000) and multi-tenant distributions per AWS account (from 20 to 500).	October 21, 2026
New `CloudFront-Viewer-TLS-Details` header	You can now use the `CloudFront-Viewer-TLS-Details` header to get the key exchange algorithm, signature scheme, and certificate public key type for the connection between the viewer and CloudFront. This header also provides the SSL/TLS version, cipher, and handshake information. You can use it to determine whether a connection used a post-quantum key exchange.	October 20, 2026
Added mutual TLS (viewer)	CloudFront supports mutual TLS (viewer).	November 24, 2025
Added log field for access logs	Added the `connection-id` field for access logs (standard logs) and real-time access logs.	November 24, 2025
Added connection logs	Added connection logs as a new logging feature for mutual TLS (viewer).	November 24, 2025

Change	Description	Date
Added Connection Functions	CloudFront supports Connection Functions for mutual TLS (viewer).	November 24, 2025
Add support for bringing your own IP to CloudFront using IPAM	CloudFront supports bringing your own IPv4 addresses through IPAM's BYOIP for global services.	November 24, 2025
AWS managed policy update	The `CloudFrontReadOnlyAccess` and `CloudFrontFullAccess` IAM policies now support `ec2:DescribeIpamPools` and `ec2:GetIpamPoolCidrs` actions.	November 24, 2025
Updates to CloudFront Functions	This release adds parameters for the origin modification helper methods in CloudFront Functions. You can use the `hostHeader`, `sni`, `allowedCertificateNames`, and `originOverrides` parameters.	November 20, 2025
Updates to CloudFront Functions	Added CBOR Web Tokens (CWT) support for CloudFront Functions.	November 20, 2025
Updates to CloudFront Functions	Added general helper methods for CloudFront Functions.	November 20, 2025

Change	Description	Date
Updated AWS managed policy	Updated `CloudFrontFullAccess` to allow permission to create an AWS WAF Web ACL resource and create, update, delete, and read access to AWS Pricing Plan Manager.	November 18, 2025
Updated AWS managed policy	Updated `CloudFrontReadOnlyAccess` to allow read-only permission to AWS Pricing Plan Manager.	November 18, 2025
Updated AWS managed policy	Updated `CloudFrontFullAccess` to allow permission to create an AWS WAF Web ACL resource and create, update, delete, and read access to AWS Pricing Plan Manager.	November 18, 2025
Updated AWS managed policy	Updated `CloudFrontReadOnlyAccess` to allow read-only permission to AWS Pricing Plan Manager.	November 18, 2025
CloudFront supports flat-rate pricing plans	You can now subscribe your distributions to a CloudFront flat-rate pricing plan.	November 18, 2025
Anycast static IPs	You can now choose between only IPv4 addresses or both IPv4 and IPv6 addresses (dualstack).	November 5, 2025

Change	Description	Date
Added support for sharing VPC origins across AWS accounts	You can create a resource share and add VPC origins to it. This lets you keep your VPC origins and CloudFront distributions in separate AWS accounts, enabling your organization to maintain multi-account requirements. Other AWS accounts can associate the shared VPC origins for their CloudFront distributions.	November 5, 2025
Added viewer security policy	Added the TLSv1.2_2025 security policy. • Supported protocols and ciphers between viewers and CloudFront • Security policy (minimum SSL/TLS version)	October 10, 2025
Added post-quantum (PQ) key exchange algorithms	Added two new PQ key exchange algorithms to CloudFront TLS policies.	September 4, 2025
IPv6 origin requests	When you use a custom origin (excluding Amazon S3 and VPC origins), you can customize origin settings for your distribution to choose how CloudFront connects to your origin using IPv4 or IPv6 addresses.	September 3, 2025

Change	Description	Date
Added new managed origin request policy	Added new managed origin request policy `HostHeaderOnly` .	August 29, 2025
CloudFront public endpoints now support IPv6	See Amazon CloudFront endpoints and quotas in the *AWS General Reference* and AWS services that support IPv6 in the *Amazon VPC User Guide*.	August 21, 2025
Added viewer security policy	Added TLSv1.3_2025, a new TLS 1.3-only security policy.	August 7, 2025
Added new origin timeout settings	Added the response completion timeout for all origins, and added the response timeout (origin read timeout) for S3 origins.	July 30, 2025
Added preconfigured standard distribution settings	Added preconfigured settings for standard distributions.	June 17, 2025
Added new console workflow for standard distribution domain setup	Added new console workflow for standard distribution domain setup.	June 17, 2025
Added example parameters	Added examples for using parameters with domain names and origin paths in distribution tenants.	June 17, 2025
Added CloudFront Functions support for CloudFront SaaS Manager	Added helper functions and the `endpoint` field for the `context` object.	May 2, 2025

Change	Description	Date
Updates to standard logging (v2)	Added the `{distributionid}` partition variable to support sending access logs to AWS Glue.	May 1, 2025
Updates to CloudFront managed policies	Added ACM permissions to the `CloudFrontReadOnlyAccess` and `CloudFrontFullAccess` managed policies.	April 28, 2025
Added support for multi-tenant distribution and distribution tenants	You can create a multi-tenant distribution to set common distribution settings based on your origin type. Then, you can reuse the multi-tenant distribution to create multiple distribution tenants that share those settings. You can then customize specific distribution tenants as you add additional websites or applications.	April 28, 2025
Updates for Lambda@Edge functions	Lambda@Edge functions now support advanced logging controls and customizing the CloudWatch log group name.	April 7, 2025
Anycast static IPs	You can use Anycast static IPs to enable routing of apex domains directly to your CloudFront distributions.	April 4, 2025

Change	Description	Date
Added additional helper methods for origin modification	Added the `selectRequestOriginById()` and `createRequestOriginGroup()` CloudFront Functions helper methods.	April 2, 2025
Updates to standard logging (v2)	Added the `{accountid}` partition variable and example suffix paths for access log delivery to Amazon S3.	February 14, 2025
Added additional real-time access log fields for standard logging (v2)	You can specify the `c-country` and `cache-behavior-path-pattern` real-time access log fields when you enable standard logging (v2).	January 31, 2025
Lambda@Edge supports newer runtime version	Lambda@Edge now supports Lambda functions with the Node.js 22 runtime.	November 22, 2024
Media quality-aware resiliency support for CloudFront	You can use the media quality-aware resiliency (MQAR) feature so that CloudFront automatically chooses the origin in an origin group with the highest media quality score.	November 21, 2024
Helper method for origin modification	Added new CloudFront Functions helper method for origin modification.	November 21, 2024

Change	Description	Date
VPC origins	Use CloudFront VPC origins to restrict access to an Application Load Balancer, Network Load Balancer, or EC2 instance origin.	November 20, 2024
Updates to managed policy	Updated managed policy `CloudFrontFullAccess` .	November 20, 2024
Anycast static IPs	You can request Anycast static IPs from CloudFront to use with your distributions.	November 20, 2024
Added support for standard logging	CloudFront supports standard logging (v2) and sending your logs to Amazon CloudWatch Logs, Amazon Data Firehose, and Amazon Simple Storage Service (Amazon S3).	November 20, 2024
Added support for gRPC	CloudFront now supports gRPC requests for your distribution.	November 20, 2024
Added new managed policy for VPC origins	Added new managed policy `AWSCloudFrontVPCOriginServiceRolePolicy` .	November 20, 2024
Lambda@Edge supports newer runtime version	Lambda@Edge now supports Lambda functions with the Python 3.13 runtime.	November 13, 2024
Evaluate with AWS Config Rules	Evaluate your CloudFront configurations with AWS Config Rules.	September 20, 2024

Change	Description	Date
Added more troubleshooting content	Added more troubleshooting content for HTTP 4xx and 5xx error response status codes.	August 26, 2024
Added new managed cache policies	Added new managed cache policies `UseOriginCacheControlHeaders` and `UseOriginCacheControlHeaders-QueryStrings` .	May 24, 2024
Added origin access control support	You can now create an origin access control (OAC) for AWS Elemental MediaPackage V2 and AWS Lambda function URL.	April 11, 2024
Real-time access log fields for CMCD	Added 18 common media client data (CMCD) fields for real-time access logs.	April 9, 2024
Getting started with a CloudFront standard distribution	Updated tutorial for a standard distribution that uses an Amazon S3 origin with origin access control (OAC).	March 18, 2024

Change	Description	Date
Code examples for CloudFront using AWS SDKs	Added code examples that show how to use CloudFront with an AWS software development kit (SDK). The examples are divided into code excerpts that show you how to call individual service functions and examples that show you how to accomplish a specific task by calling multiple functions within the same service.	February 16, 2024
AWS managed policy	The `CloudFrontReadOnlyAccess` and `CloudFrontFullAccess` IAM policies now support `KeyValueStore` operations.	December 19, 2023
JavaScript runtime 2.0	Added JavaScript runtime 2.0 features for CloudFront Functions.	November 21, 2023
CloudFront KeyValueStore	Amazon CloudFront now supports CloudFront KeyValueStore. This feature is a secure, global, low-latency key value datastore that allows read access from within CloudFront Functions. You can enable advanced customizable logic at CloudFront edge locations.	November 21, 2023

Change	Description	Date
Lambda@Edge supports newer runtime version	Lambda@Edge now supports Lambda functions with the Node.js 20 runtime.	November 15, 2023
Security dashboard	CloudFront creates a security dashboard when you create a distribution. Enable AWS WAF, manage geo restrictions, and view high-level data for requests, bots, and logs.	November 8, 2023
Sorting query strings in functions	CloudFront now supports query string sorting using CloudFront Functions.	October 3, 2023
AWS WAF security recommendations	Amazon CloudFront now displays AWS WAF security recommendations on the CloudFront console.	September 26, 2023
Support for serving stale (expired) cache content	CloudFront supports the `Stale-While-Revalidate` and `Stale-If-Error` cache control directives.	May 15, 2023
Enable AWS WAF protections with one click	A streamlined method for adding AWS WAF security protections to CloudFront distributions.	May 10, 2023
Enable ACLs for new S3 buckets used for standard logs	Added note and links to address the default ACL setting for new S3 buckets.	April 11, 2023

Change	Description	Date
Create an origin using Amazon S3 Object Lambda	You can use an Amazon S3 Object Lambda Access Point alias as an origin for your distribution.	March 31, 2023
Customize HTTP status and body using CloudFront Functions	You can use CloudFront Functions to update the viewer response status code and replace or remove the response body.	March 29, 2023
Added CORS headers wildcard options for ports	You can now include wildcard configurations for ports in CORS access-control headers.	March 20, 2023
Added new link for the AWS Security Hub CSPM User Guide	Updated language and added link to the reorganized Amazon CloudFront controls in the *AWS Security Hub CSPM User Guide*.	March 9, 2023
CloudFront now supports block lists ("all except") in origin request policies	Use block lists in origin request policies to include all query strings, HTTP headers, or cookies, ***except*** for the ones specified, in requests that CloudFront sends to the origin.	February 22, 2023
CloudFront adds a new managed origin request policy to forward all viewer headers except the Host header	Use CloudFront new managed origin request policy to include all headers from the viewer request, ***except*** for the `Host` header, in requests that CloudFront sends to the origin.	February 22, 2023

Change	Description	Date
Updated restrictions on Lambda@Edge	Lambda@Edge supports Lambda runtime managemen t configurations set to **Auto**.	February 16, 2023
Updated the IAM guidance for CloudFront	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM.	February 15, 2023
Enhanced security with origin access control	You can now secure MediaStore origins by permitting access to only the designated CloudFront distributions.	February 9, 2023
New headers for determining viewer's header structure	You can now add header order and header count to help identify the viewer based on the headers that it sends.	January 13, 2023
Lambda@Edge supports newer runtime version	Lambda@Edge now supports Lambda functions with the Node.js 18 runtime.	January 12, 2023
Remove response headers using a response headers policy	You can now use a CloudFron t response headers policy to remove headers that CloudFront received in the response from the origin. The specified headers are not included in the response that CloudFront sends to viewers.	January 3, 2023

Change	Description	Date
Continuous deployment for safely testing configuration changes	You can now deploy changes to your CDN configuration by testing with a subset of production traffic.	November 18, 2022
Release of `CloudFront-Viewer-JA3-Fingerprint` header	You can now use the JA3 fingerprint to help determine whether the request comes from a known client.	November 16, 2022
Added CORS headers wildcard options	You can now use various wildcard configurations in some CORS access-control headers.	November 11, 2022
Additional metrics for CloudFront distributions	Support for `MonitoringSubscription` in the CloudFront API and CloudFormation.	October 3, 2022
Enhanced security with origin access control	You can now secure Amazon S3 origins by permitting access to only the designated CloudFront distributions.	August 24, 2022
HTTP/3 support for CloudFront distributions	You can now choose HTTP/3 for your CloudFront distribution.	August 15, 2022
Add handshake details to CloudFront-Viewer-TLS header	You can now view information about the SSL/TLS handshake used.	June 27, 2022
New metric in Server-Timing header	Added the new `cdn-downstream-fbl` metric to `Server-Timing` headers.	June 13, 2022

Change	Description	Date
New header to get information about TLS version and cipher	You can now use the `CloudFront-Viewer-TLS` header to get information about the version of TLS (or SSL) and the cipher that was used for the connection between the viewer and CloudFront.	May 23, 2022
New FunctionThrottles metric for CloudFront Functions	With Amazon CloudWatch, you can now monitor the number of times that a CloudFront Function was throttled in a given time period.	May 4, 2022
CloudFront supports Lambda function URLs	If you build a serverless web application by using Lambda functions with function URLs, you can now add CloudFront for an array of benefits.	April 6, 2022
Server-Timing header in HTTP responses	You can now enable the `Server-Timing` header in HTTP responses sent from CloudFront to view metrics that can help you gain insights about the behavior and performance of CloudFront.	March 30, 2022

Change	Description	Date
Use AWS-managed prefix list to limit inbound traffic	You can now limit the inbound HTTP and HTTPS traffic to your origins from only the IP addresses that belong to CloudFront origin-facing servers.	February 7, 2022
New feature	CloudFront adds support for *response headers policies*, which allow you to specify the HTTP headers that CloudFront adds to HTTP responses that it sends to viewers (web browsers or other clients). You can specify the desired headers (and their values) without making any changes to the origin or writing any code. For more information, see Adding or removing HTTP headers in CloudFront responses.	November 2, 2021
New `CloudFront-Viewer-Address` request header	CloudFront adds support for a new header, `CloudFront-Viewer-Address`, that contains the IP address of the viewer that sent the HTTP request to CloudFront. For more information, see Adding CloudFront request headers.	October 25, 2021

Change	Description	Date
Lambda@Edge supports new runtime version	Lambda@Edge now supports Lambda functions with the Python 3.9 runtime. For more information, see Supported runtimes.	September 22, 2021
AWS managed policy update	CloudFront updated the **CloudFrontReadOnlyAccess** policy. For more information, see CloudFront updates to AWS managed policies.	September 8, 2021
New feature	CloudFront now supports ECDSA certificates for viewer-facing HTTPS connections. For more information, see Supported protocols and ciphers between viewers and CloudFront and Requirements for using SSL/TLS certificates with CloudFront.	July 14, 2021
New feature	CloudFront now supports more ways to move an alternate domain name from one distribution to another, without contacting Support. For more information, see Move an alternate domain name to a different distribution.	July 7, 2021

Change	Description	Date
New security policy	CloudFront now supports a new security policy, **TLSv1.2_2021**, with a smaller set of supported ciphers. For more information, see Supported protocols and ciphers between viewers and CloudFront.	June 23, 2021
New feature	Amazon CloudFront now supports CloudFront Functions, a native feature of CloudFront that enables you to write lightweight functions in JavaScript for high-scale, latency-sensitive CDN customizations. For more information, see Customizing at the edge with CloudFront Functions.	May 3, 2021
Lambda@Edge supports newer runtime versions	Lambda@Edge now supports Lambda functions with the Node.js 14 runtime. For more information, see Supported runtimes.	April 29, 2021
Remove documentation for RTMP distributions	Amazon CloudFront deprecated real-time messaging protocol (RTMP) distributions on December 31, 2020. Documentation for RTMP distributions is now removed from the *Amazon CloudFront Developer Guide*.	February 10, 2021

Change	Description	Date
New pricing option	Amazon CloudFront introduces the *CloudFront security savings bundle*, a simple way to save up to 30% on the CloudFront charges on your AWS bill. For more information, see the Savings Bundle FAQs.	February 5, 2021
New tutorial	The *Amazon CloudFront Developer Guide* now includes a tutorial for using Amazon CloudFront to restrict access to an Application Load Balancer in Elastic Load Balancing. For more information, see Restricting access to Application Load Balancers.	December 18, 2020
New option for public key management	CloudFront now supports public key management for signed URLs and signed cookies through the CloudFront console and API, without requiring access to the AWS account root user. For more information, see Specifying the signers that can create signed URLs and signed cookies.	October 22, 2020

Change	Description	Date
New feature – Origin Shield	CloudFront now supports CloudFront Origin Shield, an additional layer in the CloudFront caching infrastructure that helps to minimize your origin's load, improve its availability, and reduce its operating costs. For more information, see Using Amazon CloudFront Origin Shield.	October 20, 2020
New compression format	CloudFront now supports the Brotli compression format when you configure CloudFront to compress objects at CloudFront edge locations. You can also configure CloudFront to cache Brotli objects using a normalized `Accept-Encoding` header. For more information, see Serving compressed files and Compression support.	September 14, 2020

Change	Description	Date
New TLS protocol	CloudFront now supports the TLS 1.3 protocol for HTTPS connections between viewers and CloudFront distributions. TLS 1.3 is enabled by default in all CloudFront security policies. For more informati on, see Supported protocols and ciphers between viewers and CloudFront.	September 3, 2020
New real-time access logs	CloudFront now supports configurable real-time access logs. With real-time access logs, you can get informati on about requests made to a distribution in real time. You can use real-time access logs to monitor, analyze, and take action based on content delivery performance. For more information, see Real-time logs.	August 31, 2020
API support for additional metrics	CloudFront now supports enabling eight additional real-time metrics with the CloudFront API. For more information, see Turning on additional metrics.	August 28, 2020

Change	Description	Date
New CloudFront HTTP headers	CloudFront added additiona l HTTP headers for determini ng information about the viewer such as device type, geographic location, and more. For more informati on, see Adding CloudFront request headers.	July 23, 2020
New feature	CloudFront now supports *cache policies* and *origin request policies*, which give you more granular control over the *cache key* and *origin requests* for your CloudFron t distributions. For more information, see Control the cache key and Control origin requests.	July 22, 2020
New security policy	CloudFront now supports a new security policy, **TLSv1.2_2019**, with a smaller set of supported ciphers. For more information, see Supported protocols and ciphers between viewers and CloudFront.	July 8, 2020
New settings to control origin timeouts and attempts	CloudFront added new settings that control origin timeouts and attempts. For more information, see Controlling origin timeouts and attempts.	June 5, 2020

Change	Description	Date
New documentation for getting started with CloudFront by creating a secure static website	Get started with CloudFront by creating a secure static website using Amazon S3, CloudFront, Lambda@Edge, and more, all deployed with CloudFormation. For more information, see Getting started with a secure static website.	June 2, 2020
Lambda@Edge supports newer runtime versions	Lambda@Edge now supports Lambda functions with the Node.js 12 and Python 3.8 runtimes. For more information, see Supported runtimes.	February 27, 2020
New real-time metrics in CloudWatch	Amazon CloudFront now offers eight additional real-time metrics in Amazon CloudWatch. For more information, see Turning on additional CloudFront distribution metrics.	December 19, 2019
New fields in access logs	CloudFront adds seven new fields to access logs. For more information, see Standard log file fields.	December 12, 2019

Change	Description	Date
AWS WordPress plugin	You can use the AWS WordPress plugin to provide visitors to your WordPress website an accelerated viewing experience using CloudFront. (Update: as of September 30, 2022, the AWS for WordPress plugin is deprecated.)	October 30, 2019
Tag-based and resource-level IAM permissions policies	CloudFront now supports two additional ways of specifyin g IAM permission policies: tag-based and resource-level policy permissions. For more information, see Managing Access to Resources.	August 8, 2019
Support for Python programming language	You can now use the Python programming language to develop functions in Lambda@Edge, in addition to Node.js. For example functions that cover a variety of scenarios, see Lambda@Ed ge Example Functions.	August 1, 2019

Change	Description	Date
Updated monitoring graphs	Content updates to describe new ways for you to monitor Lambda functions associate d with your CloudFront distributions directly from the CloudFront console to more easily track and debug errors. For more information, see Monitoring CloudFront.	June 20, 2019
Consolidated security content	A new Security chapter consolidates information about CloudFront features around and implementation of data protection, IAM, logging, compliance, and more. For more information, see Security.	May 24, 2019
Domain validation is now required	CloudFront now requires that you use an SSL certificate to verify that you have permissio n to use an alternate domain name with a distribution. For more information, see Using Alternate Domain Names and HTTPS.	April 9, 2019
Updated PDF filename	The new filename for the Amazon CloudFron t Developer Guide is: AmazonCloudFront_D evGuide. The previous name was: cf-dg.	January 7, 2019

Change	Description	Date
New features	CloudFront now supports WebSocket, a TCP-based protocol that is useful when you need long-lived connections between clients and servers. You can also now set up CloudFront with origin failover for scenarios that require high availability. For more information, see Using WebSocket with CloudFront Distributions and Optimizing High Availability with CloudFront Origin Failover.	November 20, 2018
New feature	CloudFront now supports detailed error logging for HTTP requests that run Lambda functions. You can store the logs in CloudWatch and use them to help troubleshoot HTTP 5xx errors when your function returns an invalid response. For more information, see CloudWatch Metrics and CloudWatch Logs for Lambda Functions.	October 8, 2018

Change	Description	Date
New feature	You can now opt to have Lambda@Edge expose the body in a request for writable HTTP methods (POST, PUT, DELETE, and so on), so that you can access it in your Lambda function. You can choose read-only access, or you can specify that you'll replace the body. For more information, see Accessing the Request Body by Choosing the Include Body Option.	August 14, 2018
New feature	CloudFront now supports serving content compressed by using brotli or other compression algorithms, in addition to or instead of gzip. For more information, see Serving Compressed Files.	July 25, 2018
Reorganization	The Amazon CloudFront Developer Guide has been reorganized to simplify finding related content, and to improve scannability and navigation.	June 28, 2018

Change	Description	Date
New Feature	Lambda@Edge now enables you to further customize the delivery of content stored in an Amazon S3 bucket, by allowing you to access additional headers, including custom headers, within origin-facing events. For more information, see these examples showing personalization of content based on viewer location and viewer device type.	March 20, 2018
New Feature	You can now use Amazon CloudFront to negotiate HTTPS connections to origins using Elliptic Curve Digital Signature Algorithm (ECDSA). ECDSA uses smaller keys that are faster, yet, just as secure, as the older RSA algorithm . For more information, see Supported SSL/TLS Protocols and Ciphers for Communication Between CloudFront and Your Origin and About RSA and ECDSA Ciphers.	March 15, 2018

Change	Description	Date
New Feature	Lambda@Edge enables you to customize error responses from your origin, by allowing you to execute Lambda functions in response to HTTP errors that Amazon CloudFront receives from your origin. For more information, see these examples showing redirects to another location and response generation with 200 status code (OK).	December 21, 2017
New Feature	A new CloudFront capability, field-level encryption, helps you to further enhance the security of sensitive data, like credit card numbers or personally identifiable information (PII) like social security numbers. For more information, see Using field-level encryption to help protect sensitive data.	December 14, 2017
Doc history archived	Older doc history was archived.	December 1, 2017