aws

User Guide

Elastic Load Balancing

Elastic Load Balancing: User Guide

This documentation is a draft for private preview for regions in the AWS European Sovereign Cloud. Documentation content will continue to evolve. Published: January 8, 2026.

Table of Contents

What is ELB?

ELB automatically distributes your incoming traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. It monitors the health of its registered targets, and routes traffic only to the healthy targets. ELB scales your load balancer capacity automatically in response to changes in incoming traffic.

Load balancer benefits

A load balancer distributes workloads across multiple compute resources, such as virtual servers. Using a load balancer increases the availability and fault tolerance of your applications.

You can add and remove compute resources from your load balancer as your needs change, without disrupting the overall flow of requests to your applications.

You can configure health checks, which monitor the health of the compute resources, so that the load balancer sends requests only to the healthy ones. You can also offload the work of encryption and decryption to your load balancer so that your compute resources can focus on their main work.

Features of ELB

ELB supports multiple load balancer types. You can select the type of load balancer that best suits your needs. .

For more information about the current generation load balancers, see the following documentation:

- User Guide for Application Load Balancers
- User Guide for Network Load Balancers
- User Guide for Gateway Load Balancers

Classic Load Balancers are the previous generation of load balancers from ELB. We recommend that you migrate to a current generation load balancer. For more information, see Migrate your Classic Load Balancer.

Accessing ELB

You can create, access, and manage your load balancers using any of the following interfaces:

- **AWS Management Console** — Provides a web interface that you can use to access ELB.
- **AWS Command Line Interface (AWS CLI)** — Provides commands for a broad set of AWS services, including ELB. The AWS CLI is supported on Windows, macOS, and Linux. For more information, see AWS Command Line Interface.
- **AWS SDKs** — Provide language-specific APIs and take care of many of the connection details, such as calculating signatures, handling request retries, and error handling. For more information, see AWS SDKs.
- **Query API**— Provides low-level API actions that you call using HTTPS requests. Using the Query API is the most direct way to access ELB. However, the Query API requires that your application handle low-level details such as generating the hash to sign the request, and error handling. For more information, see the following:
  - Application Load Balancers, Network Load Balancers, and Gateway Load Balancers — API version 2015-12-01
  - Classic Load Balancers — API version 2012-06-01

Related services

ELB works with the following services to improve the availability and scalability of your applications.

- **Amazon EC2** — Virtual servers that run your applications in the cloud. You can configure your load balancer to route traffic to your EC2 instances. For more information, see the Amazon EC2 User Guide.
- **Amazon EC2 Auto Scaling** — Ensures that you are running your desired number of instances, even if an instance fails. Amazon EC2 Auto Scaling also enables you to automatically increase or decrease the number of instances as the demand on your instances changes. If you enable Auto Scaling with ELB, instances that are launched by Auto Scaling are automatically registered with the load balancer. Likewise, instances that are terminated by Auto Scaling are automatically de-registered from the load balancer. For more information, see the Amazon EC2 Auto Scaling User Guide.

- **AWS Certificate Manager** — When you create an HTTPS listener, you can specify certificates provided by ACM. The load balancer uses certificates to terminate connections and decrypt requests from clients.
- **Amazon CloudWatch** — Enables you to monitor your load balancer and to take action as needed. For more information, see the Amazon CloudWatch User Guide.
- **Amazon ECS** — Enables you to run, stop, and manage Docker containers on a cluster of EC2 instances. You can configure your load balancer to route traffic to your containers. For more information, see the Amazon Elastic Container Service Developer Guide.
- **AWS Global Accelerator** — Improves the availability and performance of your application. Use an accelerator to distribute traffic across multiple load balancers in one or more AWS Regions. For more information, see the AWS Global Accelerator Developer Guide.
- **Route 53** — Provides a reliable and cost-effective way to route visitors to websites by translating domain names into the numeric IP addresses that computers use to connect to each other. For example, it would translate `www.example.com` into the numeric IP address `192.0.2.1`. AWS assigns URLs to your resources, such as load balancers. However, you might want a URL that is easy for users to remember. For example, you can map your domain name to a load balancer. For more information, see the Amazon Route 53 Developer Guide.
- **AWS WAF** — You can use AWS WAF with your Application Load Balancer to allow or block requests based on the rules in a web access control list (web ACL). For more information, see the AWS WAF Developer Guide.

Pricing

With your load balancer, you pay only for what you use. For more information, see ELB pricing.

How ELB works

A load balancer accepts incoming traffic from clients and routes requests to its registered targets (such as EC2 instances) in one or more Availability Zones. The load balancer also monitors the health of its registered targets and ensures that it routes traffic only to healthy targets. When the load balancer detects an unhealthy target, it stops routing traffic to that target. It then resumes routing traffic to that target when it detects that the target is healthy again.

You configure your load balancer to accept incoming traffic by specifying one or more *listeners*. A listener is a process that checks for connection requests. It is configured with a protocol and port number for connections from clients to the load balancer. Likewise, it is configured with a protocol and port number for connections from the load balancer to the targets.

Contents

Availability Zones and load balancer nodes

When you enable an Availability Zone for your load balancer, ELB creates a load balancer node in the Availability Zone. If you register targets in an Availability Zone but do not enable the Availability Zone, these registered targets do not receive traffic. Your load balancer is most effective when you ensure that each enabled Availability Zone has at least one registered target.

We recommend enabling multiple Availability Zones for all load balancers. With an Application Load Balancer however, it is a requirement that you enable at least two or more Availability Zones. This configuration helps ensure that the load balancer can continue to route traffic. If one Availability Zone becomes unavailable or has no healthy targets, the load balancer can route traffic to the healthy targets in another Availability Zone.

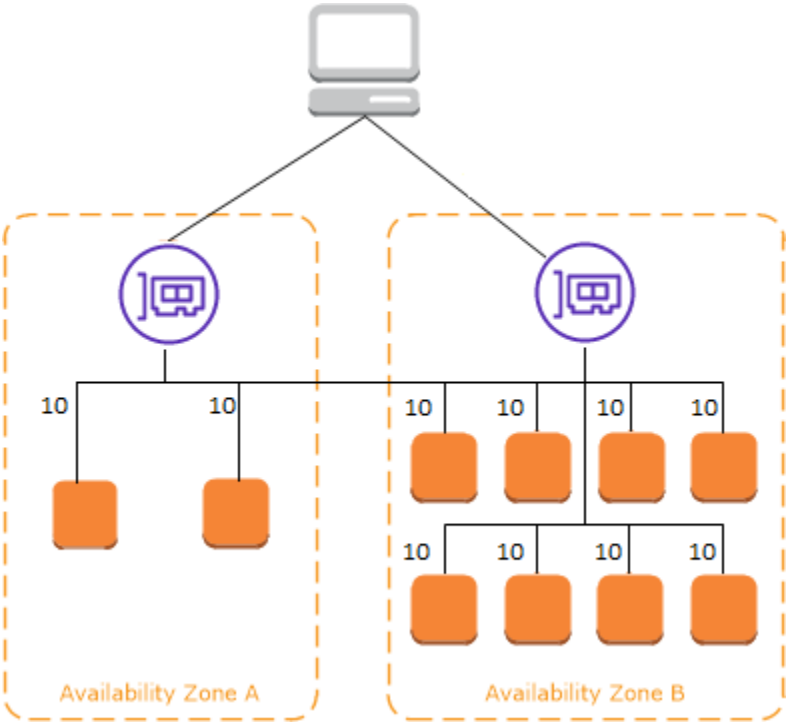
After you disable an Availability Zone, the targets in that Availability Zone remain registered with the load balancer. However, even though they remain registered, the load balancer does not route traffic to them.

Cross-zone load balancing

The nodes for your load balancer distribute requests from clients to registered targets. When cross-zone load balancing is enabled, each load balancer node distributes traffic across the registered targets in all enabled Availability Zones. When cross-zone load balancing is disabled, each load balancer node distributes traffic only across the registered targets in its Availability Zone.

The following diagrams demonstrate the effect of cross-zone load balancing with round robin as the default routing algorithm. There are two enabled Availability Zones, with two targets in Availability Zone A and eight targets in Availability Zone B. Clients send requests, and Amazon Route 53 responds to each request with the IP address of one of the load balancer nodes. Based on the round robin routing algorithm, traffic is distributed such that each load balancer node receives 50% of the traffic from the clients. Each load balancer node distributes its share of the traffic across the registered targets in its scope.

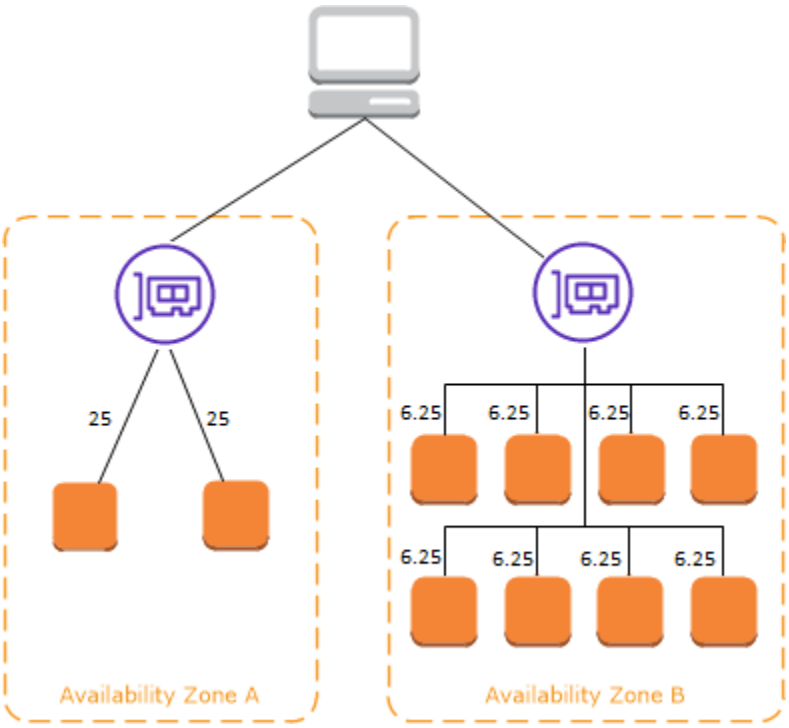
If cross-zone load balancing is enabled, each of the 10 targets receives 10% of the traffic. This is because each load balancer node can route its 50% of the client traffic to all 10 targets.



If cross-zone load balancing is disabled:

- Each of the two targets in Availability Zone A receives 25% of the traffic.
- Each of the eight targets in Availability Zone B receives 6.25% of the traffic.

This is because each load balancer node can route its 50% of the client traffic only to targets in its Availability Zone.



With Application Load Balancers, cross-zone load balancing is always enabled at the load balancer level. At the target group level, cross-zone load balancing can be disabled. For more information, see Turn off cross-zone load balancing in the *User Guide for Application Load Balancers*.

With Network Load Balancers and Gateway Load Balancers, cross-zone load balancing is disabled by default. After you create the load balancer, you can enable or disable cross-zone load balancing at any time. For more information, see Cross-zone load balancing in the *User Guide for Network Load Balancers*.

When you create a Classic Load Balancer, the default for cross-zone load balancing depends on how you create the load balancer. With the API or CLI, cross-zone load balancing is disabled by default. With the AWS Management Console, the option to enable cross-zone load balancing is selected by default. After you create a Classic Load Balancer, you can enable or disable cross-zone load balancing at any time. For more information, see Enable cross-zone load balancing in the *User Guide for Classic Load Balancers*.

Zonal shift

Zonal shift is a capability in Amazon Application Recovery Controller (ARC) (ARC). With zonal shift, you can shift a load balancer resource away from an impaired Availability Zone with a single action. This way, you can continue operating from other healthy Availability Zones in an AWS Region.

When you start a zonal shift, your load balancer stops sending traffic for the resource to the affected Availability Zone. ARC creates the zonal shift immediately. However, it can take a short time, typically up to a few minutes, to complete existing, in-progress connections in the affected Availability Zone. For more information, see How a zonal shift works: health checks and zonal IP addresses in the *Amazon Application Recovery Controller (ARC) Developer Guide*.

Before you use a zonal shift, review the following:

- Zonal shift is supported when you use a Network Load Balancer with cross-zone load balancing turned on or off.
- You can start a zonal shift for a specific load balancer only for a single Availability Zone. You can't start a zonal shift for multiple Availability Zones.
- AWS proactively removes zonal load balancer IP addresses from DNS when multiple infrastructure issues impact services. Always check current Availability Zone capacity before you start a zonal shift. If your load balancers have cross-zone load balancing turned off and you use a zonal shift to remove a zonal load balancer IP address, the Availability Zone affected by the zonal shift also loses target capacity.

For more guidance and information, see Best practices for zonal shifts in ARC in the *Amazon Application Recovery Controller (ARC) Developer Guide*.

Request routing

Before a client sends a request to your load balancer, it resolves the load balancer's domain name using a Domain Name System (DNS) server. The DNS entry is controlled by Amazon, because your load balancers are in the `amazonaws.com` domain. The Amazon DNS servers return one or more IP addresses to the client. These are the IP addresses of the load balancer nodes for your load balancer. With Network Load Balancers, ELB creates a network interface for each Availability Zone that you enable, and uses it to get a static IP address. You can optionally associate one Elastic IP address with each network interface when you create the Network Load Balancer.

As traffic to your application changes over time, ELB scales your load balancer and updates the DNS entry. The DNS entry also specifies the time-to-live (TTL) of 60 seconds. This helps ensure that the IP addresses can be remapped quickly in response to changing traffic.

The client determines which IP address to use to send requests to the load balancer. The load balancer node that receives the request selects a healthy registered target and sends the request to the target using its private IP address.

For more information, see Routing traffic to an ELB load balancer in the *Amazon Route 53 Developer Guide*.

Routing algorithm

With **Application Load Balancers**, the load balancer node that receives the request uses the following process:

1. Evaluates the listener rules in priority order to determine which rule to apply.
2. Selects a target from the target group for the rule action, using the routing algorithm configured for the target group. The default routing algorithm is round robin. Routing is performed independently for each target group, even when a target is registered with multiple target groups.

With **Network Load Balancers**, the load balancer node that receives the connection uses the following process:

1. Selects a target from the target group for the default rule using a flow hash algorithm. It bases the algorithm on:
   - The protocol
   - The source IP address and source port
   - The destination IP address and destination port
   - The TCP sequence number
2. Routes each individual TCP connection to a single target for the life of the connection. The TCP connections from a client have different source ports and sequence numbers, and can be routed to different targets.

With **Gateway Load Balancers**, the load balancer node that receives the connection uses a 5-tuple flow hash algorithm to select a target appliance. After a flow is established, all packets for

the same flow are consistently routed to the same target appliance. The load balancer and target appliances exchange traffic using the GENEVE protocol on port 6081.

With **Classic Load Balancers**, the load balancer node that receives the request selects a registered instance as follows:

- Uses the round robin routing algorithm for TCP listeners
- Uses the least outstanding requests routing algorithm for HTTP and HTTPS listeners

HTTP connections

Classic Load Balancers use pre-open connections, but Application Load Balancers do not. Both Classic Load Balancers and Application Load Balancers use connection multiplexing. This means that requests from multiple clients on multiple front-end connections can be routed to a given target through a single backend connection. Connection multiplexing improves latency and reduces the load on your applications. To prevent connection multiplexing, disable HTTP `keep-alive` headers by setting the `Connection: close` header in your HTTP responses.

Application Load Balancers and Classic Load Balancers support pipelined HTTP on front-end connections. They do not support pipelined HTTP on backend connections.

Application Load Balancers support the following HTTP request methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, and PATCH.

Application Load Balancers support the following protocols on front-end connections: HTTP/0.9, HTTP/1.0, HTTP/1.1, and HTTP/2. You can use HTTP/2 only with HTTPS listeners, and can send up to 128 requests in parallel using one HTTP/2 connection. Application Load Balancers also support connection upgrades from HTTP to WebSockets. However, if there is a connection upgrade, Application Load Balancer listener routing rules and AWS WAF integrations no longer apply.

Application Load Balancers use HTTP/1.1 on backend connections (load balancer to registered target) by default. However, you can use the protocol version to send the request to the targets using HTTP/2 or gRPC. For more information, see Protocol versions. The `keep-alive` header is supported on backend connections by default. For HTTP/1.0 requests from clients that do not have a host header, the load balancer generates a host header for the HTTP/1.1 requests sent on the backend connections. The host header contains the DNS name of the load balancer.

Classic Load Balancers support the following protocols on front-end connections (client to load balancer): HTTP/0.9, HTTP/1.0, and HTTP/1.1. They use HTTP/1.1 on backend connections (load

balancer to registered target). The `keep-alive` header is supported on backend connections by default. For HTTP/1.0 requests from clients that do not have a host header, the load balancer generates a host header for the HTTP/1.1 requests sent on the backend connections. The host header contains the IP address of the load balancer node.

HTTP headers

Application Load Balancers and Classic Load Balancers automatically add **X-Forwarded-For**, **X-Forwarded-Proto**, and **X-Forwarded-Port** headers to the request.

Application Load Balancers convert the hostnames in HTTP host headers to lower case before sending them to targets.

For front-end connections that use HTTP/2, the header names are in lowercase. Before the request is sent to the target using HTTP/1.1, the following header names are converted to mixed case: **X-Forwarded-For**, **X-Forwarded-Proto**, **X-Forwarded-Port**, **Host**, **X-Amzn-Trace-Id**, **Upgrade**, and **Connection**. All other header names are in lowercase.

Application Load Balancers and Classic Load Balancers honor the connection header from the incoming client request after proxying the response back to the client.

When Application Load Balancers and Classic Load Balancers using HTTP/1.1 receive an **Expect: 100-Continue** header, they immediately respond with **HTTP/1.1 100 Continue** without testing the content length header. The **Expect: 100-Continue** request header is not forwarded to its targets.

When using HTTP/2, Application Load Balancers do not support the **Expect: 100-Continue** header from client requests. The Application Load Balancer will not respond with **HTTP/2 100 Continue** or forward this header to its targets.

HTTP header limits

The following size limits for Application Load Balancers are hard limits that cannot be changed:

- Request line: 16 K
- Single header: 16 K
- Entire response header: 32 K
- Entire request header: 64 K

Load balancer scheme

When you create a load balancer, you must choose whether to make it an internal load balancer or an internet-facing load balancer.

The nodes of an internet-facing load balancer have public IP addresses. The DNS name of an internet-facing load balancer is publicly resolvable to the public IP addresses of the nodes. Therefore, internet-facing load balancers can route requests from clients over the internet.

The nodes of an internal load balancer have only private IP addresses. The DNS name of an internal load balancer is publicly resolvable to the private IP addresses of the nodes. Therefore, internal load balancers can only route requests from clients with access to the VPC for the load balancer.

Both internet-facing and internal load balancers route requests to your targets using private IP addresses. Therefore, your targets do not need public IP addresses to receive requests from an internal or an internet-facing load balancer.

If your application has multiple tiers, you can design an architecture that uses both internal and internet-facing load balancers. For example, this is true if your application uses web servers that must be connected to the internet, and application servers that are only connected to the web servers. Create an internet-facing load balancer and register the web servers with it. Create an internal load balancer and register the application servers with it. The web servers receive requests from the internet-facing load balancer and send requests for the application servers to the internal load balancer. The application servers receive requests from the internal load balancer.

IP address types

The IP address type that you specify for your load balancer determines how clients can communicate with the load balancer.

- **IPv4 only** – Clients communicate using public and private IPv4 addresses. The subnets that you select for your load balancer must have IPv4 address ranges.
- **Dualstack** – Clients communicate using public and private IPv4 and IPv6 addresses. The subnets that you select for your load balancer must have IPv4 and IPv6 address ranges.
- **Dualstack without public IPv4** – Clients communicate using public and private IPv6 addresses and private IPv4 addresses. The subnets that you select for your load balancer must have IPv4 and IPv6 address ranges. This option is not supported with the `internal` load balancer scheme.

The following table describes the IP address types supported for each load balancer type.

Load balancer type	IPv4 only	Dualstack	Dualstack without public IPv4
Application Load Balancer	Yes	Yes	Yes
Network Load Balancer	Yes	Yes	No
Gateway Load Balancer	Yes	Yes	No
Classic Load Balancer	Yes	No	No

The IP address type that you specify for your target group determines how the load balancer can communicate with targets.

- **IPv4 only** – The load balancer communicates using private IPv4 addresses. You must register targets with IPv4 addresses with an IPv4 target group.
- **IPv6 only** – The load balancer communicates using IPv6 addresses. You must register targets with IPv6 addresses with an IPv6 target group. The target group must be used with a dualstack load balancer.

The following table describes IP address types supported for each target group protocol.

Target group protocol	IPv4 only	IPv6 only	
HTTP and HTTPS	Yes	Yes	
TCP	Yes	Yes	

Target group protocol	IPv4 only	IPv6 only	
TLS	Yes	Yes	
UDP and TCP_UDP	Yes	Yes	
GENEVE	-	-	

Network MTU for your load balancer

The maximum transmission unit (MTU) determines the size, in bytes, for the largest packet that can be sent over the network. The larger the MTU of a connection, the more data that can be passed in a single packet. Ethernet frames consist of the packet, or the actual data you are sending, and the network overhead information that surrounds it. Traffic sent over an internet gateway has an MTU of 1500. This means that if a packet is over 1500 bytes, it is fragmented to be sent using multiple frames, or it is dropped if the `Don't Fragment` is set in the IP header.

The MTU size on load balancer nodes is not configurable. Jumbo frames (9001 MTU) are standard across load balancer nodes for Application Load Balancers, Network Load Balancers, and Classic Load Balancers. Gateway Load Balancers support 8500 MTU. For more information, see Maximum transmission unit (MTU) in the *User Guide for Gateway Load Balancers*.

The path MTU is the maximum packet size that is supported on the path between the originating host and the receiving host. Path MTU Discovery (PMTUD) is used to determine the path MTU between two devices. Path MTU Discovery is especially important if the client or target does not support jumbo frames.

When a host sends a packet that is larger than the MTU of the receiving host or larger than the MTU of a device along the path, the receiving host or device drops the packet, and then returns the following ICMP message: `Destination Unreachable: Fragmentation Needed and Don't Fragment was Set (Type 3, Code 4)`. This instructs the transmitting host to split the payload into multiple smaller packets, and retransmit them.

If packets larger than the MTU size of the client or target interface continue to be dropped, it is likely that Path MTU Discovery (PMTUD) is not working. To avoid this, ensure that Path MTU Discovery is working end to end, and that you have enabled jumbo frames on your clients and

targets. For more information about Path MTU Discovery and enabling jumbo frames, see Path MTU Discovery in the *Amazon EC2 User Guide*.

Getting started with ELB

ELB supports multiple load balancer types. You can select the type of load balancer that best suits your needs. .

Load balancers

- Create an Application Load Balancer
- Create a Network Load Balancer
- Create a Gateway Load Balancer

For demos of common load balancer configurations, see ELB demos.

If you have an existing Classic Load Balancer, you can migrate to an Application Load Balancer or a Network Load Balancer. For more information, see Migrate your Classic Load Balancer.

Security in Elastic Load Balancing

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The 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 .
- **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 company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using ELB. It shows you how to configure ELB to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your ELB resources.

With a Gateway Load Balancer, you are responsible for choosing and qualifying software from appliance vendors. You must trust the appliance software to inspect or modify traffic from the load balancer, which operates at the layer 3 of the Open Systems Interconnection (OSI) model, the network layer. The appliance vendors listed as ELB Partners have integrated and qualified their appliance software with AWS. You can place a higher degree of trust in the appliance software from vendors in this list. However, AWS does not guarantee the security or reliability of software from these vendors.

Contents

Data protection in Elastic Load Balancing

The AWS applies to data protection in ELB. 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 the Data Privacy FAQ.

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 ELB 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.

Encryption at rest

If you enable server-side encryption with Amazon S3-managed encryption keys (SSE-S3) for your S3 bucket for ELB access logs, ELB automatically encrypts each access log file before it is stored

in your S3 bucket. ELB also decrypts the access log files when you access them. Each log file is encrypted with a unique key, which is itself encrypted with a KMS key that is regularly rotated.

Encryption in transit

ELB simplifies the process of building secure web applications by terminating HTTPS and TLS traffic from clients at the load balancer. The load balancer performs the work of encrypting and decrypting the traffic, instead of requiring each EC2 instance to handle the work for TLS termination. When you configure a secure listener, you specify the cipher suites and protocol versions that are supported by your application, and a server certificate to install on your load balancer. You can use AWS Certificate Manager (ACM) or AWS Identity and Access Management (IAM) to manage your server certificates. Application Load Balancers support HTTPS listeners. Network Load Balancers support TLS listeners. Classic Load Balancers support both HTTPS and TLS listeners.

Identity and access management for ELB

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 ELB resources. IAM is an AWS service that you can use with no additional charge.

Contents

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in ELB.

Service user – If you use the ELB service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more ELB features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator.

Service administrator – If you're in charge of ELB resources at your company, you probably have full access to ELB. It's your job to determine which ELB features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to ELB.

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 ELB works with IAM

Before you use IAM to manage access to ELB, learn what IAM features are available to use with ELB.

IAM features you can use with ELB

IAM feature	ELB 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)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

Identity-based policies for ELB

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*.

Resource-based policies within ELB

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 ELB

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 ELB actions, see Actions defined by ELB V2 and Actions defined by ELB V1 in the *Service Authorization Reference*.

Policy actions in ELB use the following prefix before the action:

```
elasticloadbalancing
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
    "elasticloadbalancing:action1",
    "elasticloadbalancing:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "elasticloadbalancing:Describe*"
```

For the complete list of the API actions for ELB, see the following documentation:

- Application Load Balancers, Network Load Balancers, and Gateway Load Balancers — API Reference version 2015-12-01
- Classic Load Balancers — API Reference version 2012-06-01

Policy resources for ELB

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": "*"
```

Some ELB API actions support multiple resources. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
    "resource1",
    "resource2"
]
```

To see a list of ELB resource types and their ARNs, see Resources defined by ELB V2 and Resources defined by ELB V1 in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by ELB V2 and Actions defined by ELB V1.

Policy condition keys for ELB

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 ELB condition keys, see Condition keys for ELB V2 and Condition keys for ELB V1 in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by ELB V2 and Actions defined by ELB V1.

Condition keys

- elasticloadbalancing:ListenerProtocol condition key
- elasticloadbalancing:SecurityPolicy condition key
- elasticloadbalancing:Scheme condition key
- elasticloadbalancing:SecurityGroup condition key
- elasticloadbalancing:Subnet condition key
- elasticloadbalancing:ResourceTag condition key

elasticloadbalancing:ListenerProtocol condition key

The `elasticloadbalancing:ListenerProtocol` condition key can be used for conditions that define the types of listeners that can be created and used. The policy is available for Application Load Balancers, Network Load Balancers, and Classic Load Balancers. The following actions support this condition key:

API version 2015-12-01

- `CreateListener`

- `ModifyListener`

API version 2012-06-01

- `CreateLoadBalancer`
- `CreateLoadBalancerListeners`

The following example policy requires users to select the HTTPS protocol for the listeners for their Application Load Balancers and the TLS protocol for the listeners for their Network Load Balancers.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:CreateListener",
            "elasticloadbalancing:ModifyListener"
        ],
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "elasticloadbalancing:ListenerProtocol": [
                    "HTTPS",
                    "TLS"
                ]
            }
        }
    }
}
```

With a Classic Load Balancer, you can specify multiple listeners in a single call. Therefore, your policy must use a multi-value context key, as shown in the following example.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateLoadBalancerListeners"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "elasticloadbalancing:ListenerProtocol": [
                        "TCP",
                        "HTTP",
                        "HTTPS"
                    ]
                }
            }
        }
    ]
}
```

elasticloadbalancing:SecurityPolicy condition key

The `elasticloadbalancing:SecurityPolicy` condition key can be used for conditions that define and enforce specific security policies on the load balancers. The policy is available for Application Load Balancers, Network Load Balancers and Classic Load Balancers. The following actions support this condition key:

API version 2015-12-01

- `CreateListener`
- `ModifyListener`

API version 2012-06-01

- `CreateLoadBalancerPolicy`
- `SetLoadBalancerPoliciesOfListener`

The following example policy requires users to select one of the specified security policies for their Application Load Balancers and Network Load Balancers.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:CreateListener",
            "elasticloadbalancing:ModifyListener"
        ],
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "elasticloadbalancing:SecurityPolicy": [
                    "ELBSecurityPolicy-TLS13-1-2-2021-06",
                    "ELBSecurityPolicy-TLS13-1-2-Res-2021-06",
                    "ELBSEcurityPolicy-TLS13-1-1-2021-06"
                ]
            }
        }
    }
}
```

elasticloadbalancing:Scheme condition key

The `elasticloadbalancing:Scheme` condition key can be used for conditions that define which scheme can be selected during load balancer creation. The policy is available for Application Load Balancers, Network Load Balancers, and Classic Load Balancers. The following actions support this condition key:

API version 2015-12-01

- `CreateLoadBalancer`

API version 2012-06-01

- `CreateLoadBalancer`

The following example policy requires users to select the specified scheme for their load balancers.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "elasticloadbalancing:CreateLoadBalancer",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "elasticloadbalancing:Scheme": "internal"
            }
        }
    }
}
```

`elasticloadbalancing:SecurityGroup` condition key

> **Important**
>
> ELB accepts all capitalizations of security group IDs. However, make sure to use the appropriate case insensitive condition operators, for example `StringEqualsIgnoreCase`.

The `elasticloadbalancing:SecurityGroup` condition key can be used for conditions that define which security groups can be applied to the load balancers. The policy is available for Application Load Balancers, Network Load Balancers and Classic Load Balancers. The following actions support this condition key:

API version 2015-12-01

- `CreateLoadBalancer`
- `SetSecurityGroups`

API version 2012-06-01

- `CreateLoadBalancer`

- `ApplySecurityGroupsToLoadBalancer`

The following example policy requires users to select one of the specified security groups for their load balancers.

```
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:CreateLoadBalancer",
            "elasticloadbalancing:SetSecurityGroup"
        ],
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEqualsIgnoreCase":{
                "elasticloadbalancing:SecurityGroup": [
                    "sg-51530134",
                    "sg-51530144",
                    "sg-51530139"
                ]
            },
        }
    }
}
```

elasticloadbalancing:Subnet condition key

> **Important**
>
> ELB accepts all capitalizations of subnet IDs. However, make sure to use the appropriate case insensitive condition operators, for example `StringEqualsIgnoreCase`.

The `elasticloadbalancing:Subnet` condition key can be used for conditions that define which subnets can be created and attached to load balancers. The policy is available for Application Load Balancers, Network Load Balancers, Gateway Load Balancers and Classic Load Balancers. The following actions support this condition key:

API version 2015-12-01

- `CreateLoadBalancer`
- `SetSubnets`

API version 2012-06-01

- `CreateLoadBalancer`
- `AttachLoadBalancerToSubnets`

The following example policy requires users to select one of the specified subnets for their load balancers.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:CreateLoadBalancer",
            "elasticloadbalancing:SetSubnets"
        ],
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEqualsIgnoreCase": {
                "elasticloadbalancing:Subnet": [
                    "subnet-01234567890abcdef",
                    "subnet-01234567890abcdeg "
                ]
            }
        }
    }
}
```

elasticloadbalancing:ResourceTag condition key

The `elasticloadbalancing:ResourceTag/`*`key`* condition key is specific to ELB. All mutating actions support this condition key.

ACLs in ELB

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 ELB

Supports ABAC (tags in policies): Yes

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*.

Using temporary credentials with ELB

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*.

Cross-service principal permissions for ELB

Supports forward access sessions (FAS): Yes

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 ELB

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*.

Service-linked roles for ELB

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.

For details about creating or managing ELB service-linked roles, see ELB service-linked role.

ELB API permissions to tag resources during creation

For users to tag resources during creation, they must have permissions to use the action that creates the resource, such as `elasticloadbalancing:CreateLoadBalancer` or `elasticloadbalancing:CreateTargetGroup`. If tags are specified in the resource-creating action, additional authorization is required on the `elasticloadbalancing:AddTags` action to verify if users have permissions to apply tags to the resources being created. Therefore, users must also have explicit permissions to use the `elasticloadbalancing:AddTags` action.

In the IAM policy definition for the `elasticloadbalancing:AddTags` action, you can use the `Condition` element with the `elasticloadbalancing:CreateAction` condition key to give tagging permissions to the action that creates the resource.

The following example demonstrates a policy that allows users to create target groups and apply any tags to them during creation. Users are not permitted to tag any existing resources (they can't call the `elasticloadbalancing:AddTags` action directly).

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "elasticloadbalancing:CreateTargetGroup"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
         "elasticloadbalancing:AddTags"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
             "elasticloadbalancing:CreateAction" : "CreateTargetGroup"
         }
      }
    }
  ]
}
```

Similarly, the following policy allows users to create a load balancer and apply tags during creation. Users are not permitted to tag any existing resources (they can't call the `elasticloadbalancing:AddTags` action directly).

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
         "elasticloadbalancing:CreateLoadBalancer"
```

```
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
         "elasticloadbalancing:AddTags"
      ],
      "Resource": "*",
      "Condition": {
         "StringEquals": {
             "elasticloadbalancing:CreateAction" : "CreateLoadBalancer"
         }
      }
    }
  ]
}
```

The `elasticloadbalancing:AddTags` action is only evaluated if tags are applied during the resource-creating action. Therefore, a user that has permissions to create a resource (assuming there are no tagging conditions) does not require permissions to use the `elasticloadbalancing:AddTags` action if no tags are specified in the request. However, if the user attempts to create a resource with tags, the request fails if the user does not have permissions to use the `elasticloadbalancing:AddTags` action.

ELB service-linked role

ELB uses a service-linked role for the permissions that it requires to call other AWS services on your behalf. For more information, see Service-linked roles in the *IAM User Guide*.

Permissions granted by the service-linked role

ELB uses the service-linked role named AWSServiceRoleForElasticLoadBalancing to call other AWS services on your behalf.

AWSServiceRoleForElasticLoadBalancing trusts the `elasticloadbalancing.amazonaws.com` service to assume the role.

The role permissions policy is AWSElasticLoadBalancingServiceRolePolicy. To view the permissions for this policy, see AWSElasticLoadBalancingServiceRolePolicy in the *AWS Managed Policy Reference*.

Create the service-linked role

You don't need to manually create the AWSServiceRoleForElasticLoadBalancing role. ELB creates this role for you when you create a load balancer or a target group.

For ELB to create a service-linked role on your behalf, you must have the required permissions. For more information, see Service-linked role permissions in the *IAM User Guide*.

Edit the service-linked role

You can edit the description of AWSServiceRoleForElasticLoadBalancing using IAM. For more information, see Edit a service-linked role description in the *IAM User Guide*.

Delete the service-linked role

If you no longer need to use ELB, we recommend that you delete AWSServiceRoleForElasticLoadBalancing.

You can delete this service-linked role only after you delete all load balancers in your AWS account. This ensures that you can't inadvertently remove permission to access your load balancers. For more information, see Delete an Application Load Balancer, Delete a Network Load Balancer, and Delete a Classic Load Balancer.

You can use the IAM console, the IAM CLI, or the IAM API to delete service-linked roles. For more information, see Delete a service-linked role in the *IAM User Guide*.

After you delete AWSServiceRoleForElasticLoadBalancing, ELB creates the role again if you create a load balancer.

AWS managed policies for ELB

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AWSElasticLoadBalancingClassicServiceRolePolicy

This policy includes all the permissions that ELB (Classic Load Balancer) requires to call other AWS services on your behalf. Service-linked roles are predefined. With predefined roles you don't have to manually add the necessary permissions for ELB to complete actions on your behalf. You cannot attach, detach, modify, or delete this policy.

To view the permissions for this policy, see AWSElasticLoadBalancingClassicServiceRolePolicy in the *AWS Managed Policy Reference*.

AWS managed policy: AWSElasticLoadBalancingServiceRolePolicy

This policy includes all the permissions that ELB requires to call other AWS services on your behalf. Service-linked roles are predefined. With predefined roles you don't have to manually add the necessary permissions for ELB to complete actions on your behalf. You cannot attach, detach, modify, or delete this policy.

To view the permissions for this policy, see AWSElasticLoadBalancingServiceRolePolicy in the *AWS Managed Policy Reference*.

AWS managed policy: ElasticLoadBalancingFullAccess

This policy gives full access to the ELB service and limited access to other services via the AWS Management Console.

To view the permissions for this policy, see ElasticLoadBalancingFullAccess in the *AWS Managed Policy Reference*.

AWS managed policy: ElasticLoadBalancingReadOnly

This policy provides read-only access to ELB and dependent services.

To view the permissions for this policy, see ElasticLoadBalancingReadOnly in the *AWS Managed Policy Reference*.

ELB updates to AWS managed policies

View details about updates to AWS managed policies for ELB since this service began tracking these changes.

Change	Description	Date
AWSElasticLoadBalancingServiceRolePolicy - Update to an existing policy	Added the `ec2:AllocateIpamPoolCidr` action to grant permissions to allocate CIDR blocks from IPAM pools.	February 17, 2025
ElasticLoadBalancingFullAccess - Update to an existing policy	Added the `arc-zonal-shift:*` actions to grant permissions required for zonal shift.	November 28, 2023
ElasticLoadBalancingReadOnly - Update to an existing policy	Added the following actions to grant permissions required for zonal shift: `arc-zonal-shift:GetManagedResource`, `arc-zonal-shift:ListManagedResources` and `arc-zonal-shift:ListZonalShifts`.	November 28, 2023
AWSElasticLoadBalancingServiceRolePolicy - Update to an existing policy	Added the `ec2:DescribeVpcPeeringConnections` action to grant permissions required for peering connections.	October 11, 2021
ElasticLoadBalancingFullAccess - Update to an existing policy	Added the `ec2:DescribeVpcPeeringConnections` action to grant permissions required for peering connections.	October 11, 2021
ElasticLoadBalancingFullAccess - New policy	Provides full access to ELB and dependent services.	September 20, 2018
ElasticLoadBalancingReadOnly - New policy	Provides read-only access to ELB and dependent services.	September 20, 2018

Change	Description	Date
ELB started tracking changes	ELB started tracking changes for its AWS managed policies.	September 20, 2018

Compliance validation for Elastic Load Balancing

To learn whether an AWS service is within the scope of specific compliance programs, see and choose the compliance program that you are interested in. For general information, see .

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in Elastic Load Balancing

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between 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.

In addition to the AWS Global Infrastructure, ELB provides the following features to support your data resiliency:

- Distributes incoming traffic across multiple instances in a single Availability Zone or multiple Availability Zones.
- You can use AWS Global Accelerator with your Application Load Balancers to distribute incoming traffic across multiple load balancers in one or more AWS Regions. For more information, see the AWS Global Accelerator Developer Guide.
- Amazon ECS enables you to run, stop, and manage Docker containers on a cluster of EC2 instances. You can configure your Amazon ECS service to use a load balancer to distribute

incoming traffic across the services in a cluster. For more information, see the Amazon Elastic Container Service Developer Guide.

Infrastructure security in Elastic Load Balancing

As a managed service, ELB 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 ELB 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.

Network isolation

A virtual private cloud (VPC) is a virtual network in your own logically isolated area in the AWS Cloud. A subnet is a range of IP addresses in a VPC. When you create a load balancer, you can specify one or more subnets for the load balancer nodes. You can deploy EC2 instances in the subnets of your VPC and register them with your load balancer. For more information about VPC and subnets, see the Amazon VPC User Guide.

When you create a load balancer in a VPC, it can be either internet-facing or internal. An internal load balancer can only route requests that come from clients with access to the VPC for the load balancer.

Your load balancer sends requests to its registered targets using private IP addresses. Therefore, your targets do not need public IP addresses in order to receive requests from a load balancer.

To call the ELB API from your VPC using private IP addresses, use AWS PrivateLink. For more information, see Access ELB using an interface endpoint (AWS PrivateLink).

Controlling network traffic

Consider the following options for securing network traffic when you use a load balancer:

- Use secure listeners to support encrypted communication between clients and your load balancers. Application Load Balancers support HTTPS listeners. Network Load Balancers support TLS listeners. Classic Load Balancers support both HTTPS and TLS listeners. You can choose from predefined security policies for your load balancer to specify the cipher suites and protocol versions that are supported by your application. You can use AWS Certificate Manager (ACM) or AWS Identity and Access Management (IAM) to manage the server certificates installed on your load balancer. You can use the Server Name Indication (SNI) protocol to serve multiple secure websites using a single secure listener. SNI is automatically enabled for your load balancer when you associate more than one server certificate with a secure listener.
- Configure the security groups for your Application Load Balancers and Classic Load Balancers to accept traffic only from specific clients. These security groups must allow inbound traffic from clients on the listener ports and outbound traffic to the clients.
- Configure the security groups for your Amazon EC2 instances to accept traffic only from the load balancer. These security groups must allow inbound traffic from the load balancer on the listener ports and the health check ports.
- Configure your Application Load Balancer to securely authenticate users through an identity provider or using corporate identities. For more information, see Authenticate users using an Application Load Balancer.
- Use AWS WAF with your Application Load Balancers to allow or block requests based on the rules in a web access control list (web ACL).

Access ELB using an interface endpoint (AWS PrivateLink)

You can establish a private connection between your virtual private cloud (VPC) and the ELB API by creating an interface VPC endpoint. You can use this connection to call the ELB API from your VPC without requiring that you attach an internet gateway, NAT instance, or VPN connection to your VPC. The endpoint provides reliable, scalable connectivity to the ELB API, versions 2015-12-01 and 2012-06-01, which you use to create and manage your load balancers.

Interface VPC endpoints are powered by AWS PrivateLink, a feature that enables communication between your applications and AWS services using private IP addresses. For more information, see AWS PrivateLink.

Limit

AWS PrivateLink does not support Network Load Balancers with more than 50 listeners.

Create an interface endpoint for ELB

Create an endpoint for ELB using the following service name:

```
com.amazonaws.region.elasticloadbalancing
```

For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create a VPC endpoint policy for ELB

You can attach a policy to your VPC endpoint to control access to the ELB API. The policy specifies:

- The principal that can perform actions.
- The actions that can be performed.
- The resource on which the actions can be performed.

The following example shows a VPC endpoint policy that denies everyone permission to create a load balancer through the endpoint. The example policy also grants everyone permission to perform all other actions.

```
{
   "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "elasticloadbalancing:CreateLoadBalancer",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Request throttling for the ELB API

ELB throttles its API requests for each AWS account on a per-Region basis. We do this to help the performance and availability of the service. Throttling ensures that requests to the ELB API do not exceed the maximum allowed API request limits. API requests are subject to the request limits whether you call them or they are called on your behalf (for example, by the AWS Management Console or a third-party application).

If you exceed an ELB API throttling limit, you get the `ThrottlingException` error code and a `Rate exceeded` error message.

We recommend that you prepare to handle throttling gracefully. For more information, see Timeouts, retries, and backoff with jitter. If you experience a high level of throttling, you can contact AWS Support to help you evaluate your API usage and potential solutions. Each case is evaluated individually. Support might increase your limits within the safety limits of the system, to maintain high availability and predictable performance.

How throttling is applied

ELB uses the token bucket algorithm to implement API throttling. With this algorithm, your account has a *bucket* that holds a specific number of *tokens*. The number of tokens in the bucket represents your throttling limit at any given second.

ELB provides two sets of API actions. ELB API version 2 supports the following types of load balancers: Application Load Balancers, Network Load Balancers, and Gateway Load Balancers. ELB API version 1 supports Classic Load Balancers. Each ELB API version has its own buckets and tokens.

Services that call the ELB API on your behalf, such as Amazon EC2, Amazon ECS, Amazon EC2 Auto Scaling, and AWS CloudFormation have their own account-level buckets. These services do not consume tokens from your buckets.

Request rate limiting

With request rate limiting, you are throttled on the number of API requests that you make. Each request that you make removes one token from the bucket. For example, the token bucket size for non-mutating API actions is 40 tokens. You can make up to 40 `Describe*` requests in one second. If you exceed 40 `Describe*` requests in one second, you are throttled and the remaining requests within that second fail.

Buckets automatically refill at a set rate. If a bucket is below its maximum capacity, a set number of tokens is added back every second until the bucket reaches its maximum capacity. If a bucket is full when refill tokens arrive, they are discarded. A bucket can't hold more than its maximum number of tokens. For example, the bucket size for non-mutating API actions is 40 tokens and the refill rate is 10 tokens per second. If you make 40 `DescribeLoadBalancers` requests in one second, the bucket is reduced to zero (0) tokens. We add 10 refill tokens to the bucket every second, until it reaches its maximum capacity of 40 tokens. This means that it takes 4 seconds for an empty bucket to reach its maximum capacity, if no requests are made during that time.

You do not need to wait for a bucket to be completely full before you can make API requests. You can use tokens as they are added to a bucket. If you immediately use the refill tokens, the bucket does not reach its maximum capacity.

There is an account-level throttling limit that is shared across all ELB API actions. The capacity of the account-level bucket is 40 tokens and the refill rate is 10 request tokens per second.

Request token bucket sizes and refill rates

For request rate limiting purposes, API actions are grouped into categories. Each category has its own limits.

Categories

- **Mutating actions** — API actions that create, modify, or delete resources. This category generally includes all API actions that are not categorized as *non-mutating actions*. These actions have a lower throttling limit than non-mutating API actions.
- **Non-mutating actions** — API actions that retrieve data about resources. These API actions typically have the highest API throttling limits.
- **Resource-intensive actions** — API actions that take the most time and consume the most resources to complete. These actions have an even lower throttling limit than mutating actions. These actions are throttled separately from other mutating actions.
- **Registration actions** — API actions that register or deregister targets. These API actions are throttled separately from other mutating actions.
- **Uncategorized actions** — These API actions receive their own token bucket sizes and refill rates, even though they fall under one of the other categories.

The following table shows the default capacity and refill rates for the categorized request token buckets.

Category	ELBv2 actions	ELBv1 actions	Bucket capacity	Refill rate (per second)
Resource-intensive	`CreateLoadBalancer` , `SetSubnets`	`CreateLoadBalancer` , `AttachLoadBalancerToSubnets` , `DetachLoadBalancerFromSubnets` , `EnableAvailabilityZonesForLoadBalancer` , `DisableAvailabilityZonesForLoadBalancer`	10	0.2 †
Registration	`RegisterTargets` , `DeregisterTargets`	`RegisterInstancesWithLoadBalancer` , `DeregisterInstancesFromLoadBalancer`	20	4
Non-mutating	`DescribeAccountLimits` , `DescribeCapacityReservation` , `DescribeListenerAttributes` , `DescribeListenerCertificates` , `DescribeListeners` , `DescribeLoadBalancerAttributes` , `DescribeLoadBalancers` , `DescribeRules` , `DescribeSSLPolicies` , `DescribeTags` ,	`Describe*`	40	10

Category	ELBv2 actions	ELBv1 actions	Bucket capacity	Refill rate (per second)
	DescribeTargetGroupAttributes , DescribeTargetGroups ,DescribeTargetHealth			
Mutating	AddListenerCertificates ,AddTags, CreateListener , CreateRule ,CreateTargetGroup ,DeleteListener ,DeleteLoadBalancer ,DeleteRule ,DeleteTargetGroup , ModifyCapacityReservation ,ModifyIpPools ,ModifyListener , ModifyListenerAttributes ,ModifyLoadBalancerAttributes ,ModifyRule , ModifyTargetGroup , ModifyTargetGroupAttributes ,RemoveListenerCertificates , RemoveTags ,SetIpAddressType ,SetRulePriorities ,SetSecurityGroups	AddTags, ApplySecurityGroupsToLoadBalancer ,Configure HealthCheck , CreateAppCookieStickinessPolicy , CreateLbCookieStickinessPolicy , CreateLoadBalancer Listener ,CreateLoadBalancerPolicy , Delete*, ModifyLoadBalancerAttributes ,RemoveTags , SetLoadBalancer*	20	3

The following table shows the default capacity and refill rates for the uncategorized request token buckets for ELBv2.

ELBv2 actions	Bucket capacity	Refill rate (per second)
`CreateTrustStore`	10	0.2 †
`AddTrustStoreRevocations` , `DeleteSharedTrustStoreAssociation` , `DeleteTrustStore` , `ModifyTrustStore` , `RemoveTrustStoreRevocations`	10	0.2 †
`GetResourcePolicy` , `GetTrustStoreCaCertificatesBundle` , `GetTrustStoreRevocationContent`	20	4
`DescribeTrustStoreAssociations` , `DescribeTrustStoreRevocations` , `DescribeTrustStores`	40	10

† Fractional refill rates require several seconds to generate one full token.

Monitoring API requests

You can use AWS CloudTrail to monitor your ELB API requests. For more information, see Log API calls for ELB using AWS CloudTrail.

Understand codes for ELB in billing and usage reports

When you use ELB, we include related codes in your AWS billing and usage reports. Reviewing these codes helps you understand your load balancer costs and usage patterns. Tracking and managing your expenses is essential for optimizing your costs.

For more information, see ELB pricing.

The following tables describe the codes for ELB that appear in your billing and usage reports. The units are hours or load balancer capacity units (LCU). Each load balancer type has a specific definition of LCU. For information about the LCUs for each load balancer type, see ELB pricing. For a list of the Region codes used in the billing and usage reports, see AWS Region billing codes.

Application Load Balancers

Code	Description	Units
region-LoadBalancerUsage	The running time.	Hours
region-LCUUsage	The LCUs used.	LCU
region-IdleProvisionedLBCapacity	The LCUs reserved but not used.	LCU
region-TS-LoadBalancerUsage	The time that a trust store is used by Mutual TLS.	Hours
region-Outposts-LoadBalancerUsage	The running time on Outposts.	Hours
region-Outposts-LCUUsage	The LCUs used on Outposts.	LCU
region-ReservedLCUUsage	The LCUs reserved.	LCU

Network Load Balancers

Code	Description	Units
region-LoadBalancerUsage	The running time.	Hours
region-LCUUsage	The LCUs used.	LCU

Gateway Load Balancers

Code	Description	Units
region-LoadBalancerUsage	The running time.	Hours
region-LCUUsage	The LCUs used.	LCU

Classic Load Balancers

Code	Description	Units
region-LoadBalancerUsage	The running time.	Hours
region-DataProcessing-Bytes	The data processed.	GB
region-IdleProvisionedLBCapacity	The LCUs reserved but not used.	LCU

Log API calls for ELB using AWS CloudTrail

ELB is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service. CloudTrail captures API calls for ELB as events. The calls captured include calls from the AWS Management Console and code calls to the ELB API operations. Using the information collected by CloudTrail, you can determine the request that was made to ELB, 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.

ELB 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.

ELB logs control plane operations as management events. For a list of the control plane operations, see the following:

- Application Load Balancers — Elastic Load Balancing API Reference version 2015-12-01
- Network Load Balancers — Elastic Load Balancing API Reference version 2015-12-01
- Gateway Load Balancers — Elastic Load Balancing API Reference version 2015-12-01
- Classic Load Balancers — Elastic Load Balancing API Reference version 2012-06-01

ELB 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.

The following examples show CloudTrail events for a user who created a load balancer and then deleted it using the AWS CLI. You can identify the CLI using the `userAgent` elements. You can identify the requested API calls using the `eventName` elements. Information about the user (`Alice`) can be found in the `userIdentity` element.

Example Example 1: CreateLoadBalancer from the ELBv2 API


```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alice"
    },
    "eventTime": "2016-04-01T15:31:48Z",
    "eventSource": "elasticloadbalancing.amazonaws.com",
    "eventName": "CreateLoadBalancer",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "198.51.100.1",
    "userAgent": "aws-cli/1.10.10 Python/2.7.9 Windows/7 botocore/1.4.1",
    "requestParameters": {
        "subnets": ["subnet-8360a9e7","subnet-b7d581c0"],
        "securityGroups": ["sg-5943793c"],
        "name": "my-load-balancer",
        "scheme": "internet-facing"
    },
    "responseElements": {
        "loadBalancers":[{
            "type": "application",
            "loadBalancerName": "my-load-balancer",
            "vpcId": "vpc-3ac0fb5f",
            "securityGroups": ["sg-5943793c"],
            "state": {"code":"provisioning"},
            "availabilityZones": [
               {"subnetId":"subnet-8360a9e7","zoneName":"us-west-2a"},
               {"subnetId":"subnet-b7d581c0","zoneName":"us-west-2b"}
            ],
```

```
            "dNSName": "my-load-balancer-1836718677.us-west-2.elb.amazonaws.com",
            "canonicalHostedZoneId": "Z2P70J7HTTTPLU",
            "createdTime": "Apr 11, 2016 5:23:50 PM",
            "loadBalancerArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:loadbalancer/app/my-load-balancer/ffcddace1759e1d0",
            "scheme": "internet-facing"
        }]
    },
    "requestID": "b9960276-b9b2-11e3-8a13-f1ef1EXAMPLE",
    "eventID": "6f4ab5bd-2daa-4d00-be14-d92efEXAMPLE",
    "eventType": "AwsApiCall",
    "apiVersion": "2015-12-01",
    "recipientAccountId": "123456789012"
}
```

Example Example 2: DeleteLoadBalancer from the ELBv2 API

```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alice"
    },
    "eventTime": "2016-04-01T15:31:48Z",
    "eventSource": "elasticloadbalancing.amazonaws.com",
    "eventName": "DeleteLoadBalancer",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "198.51.100.1",
    "userAgent": "aws-cli/1.10.10 Python/2.7.9 Windows/7 botocore/1.4.1",
    "requestParameters": {
        "loadBalancerArn": "arn:aws:elasticloadbalancing:us-
west-2:123456789012:loadbalancer/app/my-load-balancer/ffcddace1759e1d0"
    },
    "responseElements": null,
    "requestID": "349598b3-000e-11e6-a82b-298133eEXAMPLE",
    "eventID": "75e81c95-4012-421f-a0cf-babdaEXAMPLE",
    "eventType": "AwsApiCall",
    "apiVersion": "2015-12-01",
    "recipientAccountId": "123456789012"
```

```
}
```

Example Example 3: CreateLoadBalancer from the ELB API

```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAJDPLRKLG7UEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alice"
    },
    "eventTime": "2016-04-01T15:31:48Z",
    "eventSource": "elasticloadbalancing.amazonaws.com",
    "eventName": "CreateLoadBalancer",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "198.51.100.1",
    "userAgent": "aws-cli/1.10.10 Python/2.7.9 Windows/7 botocore/1.4.1",
    "requestParameters": {
        "subnets": ["subnet-12345678","subnet-76543210"],
        "loadBalancerName": "my-load-balancer",
        "listeners": [{
           "protocol": "HTTP",
           "loadBalancerPort": 80,
           "instanceProtocol": "HTTP",
           "instancePort": 80
        }]
    },
    "responseElements": {
        "dNSName": "my-loadbalancer-1234567890.elb.amazonaws.com"
    },
    "requestID": "b9960276-b9b2-11e3-8a13-f1ef1EXAMPLE",
    "eventID": "6f4ab5bd-2daa-4d00-be14-d92efEXAMPLE",
    "eventType": "AwsApiCall",
    "apiVersion": "2012-06-01",
    "recipientAccountId": "123456789012"
}
```

Example Example 4: DeleteLoadBalancer from the ELB API

```
{
```

```
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAJDPLRKLG7UEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Alice"
    },
    "eventTime": "2016-04-08T12:39:25Z",
    "eventSource": "elasticloadbalancing.amazonaws.com",
    "eventName": "DeleteLoadBalancer",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "198.51.100.1",
    "userAgent": "aws-cli/1.10.10 Python/2.7.9 Windows/7 botocore/1.4.1",
    "requestParameters": {
        "loadBalancerName": "my-load-balancer"
    },
    "responseElements": null,
    "requestID": "f0f17bb6-b9ba-11e3-9b20-999fdEXAMPLE",
    "eventID": "4f99f0e8-5cf8-4c30-b6da-3b69fEXAMPLE"
    "eventType": "AwsApiCall",
    "apiVersion": "2012-06-01",
    "recipientAccountId": "123456789012"
}
```


For information about CloudTrail record contents, see CloudTrail record contents in the *AWS CloudTrail User Guide*.

Migrate your Classic Load Balancer

Elastic Load Balancing supports the following types of load balancers: Application Load Balancers, Network Load Balancers, Gateway Load Balancers, and Classic Load Balancers. For information about the different features of each load balancer type, see ELB features.

You can also choose to migrate an existing Classic Load Balancer in a VPC, to an Application Load Balancer or a Network Load Balancer.

Benefits of migrating from a Classic Load Balancer

Each type of load balancer has its own unique features, functions, and configurations. Review the benefits of each load balancer to help decide which one is best for you.

Application Load Balancer

Using an Application Load Balancer instead of a Classic Load Balancer has the following benefits:

Support for:

- Path conditions, Host conditions, and HTTP header conditions.
- Redirecting requests from one URL to another, and routing requests to multiple applications on a single EC2 instance.
- Returning custom HTTP responses.
- Registering targets by IP address, and registering Lambda functions as targets. Including targets outside the VPC for the load balancer.
- Authenticating users through corporate or social identities.
- Amazon Elastic Container Service (Amazon ECS) containerized applications.
- Independently monitoring the health of each service.

Access logs contain additional information and are stored in a compressed format.

Improved load balancer performance overall.

Network Load Balancer

Using a Network Load Balancer instead of a Classic Load Balancer has the following benefits:

Support for:

- Static IP addresses, which allow assigning one Elastic IP address per subnet enabled for the load balancer.
- Registering targets by IP address, including targets outside the VPC for the load balancer.
- Routing requests to multiple applications on a single EC2 instance.
- Amazon Elastic Container Service (Amazon ECS) containerized applications.
- Independently monitoring the health of each service.

Ability to handle volatile workloads and scale to millions of requests per second.

Migrate using migration wizard

Migration wizard uses the configuration of your Classic Load Balancer to create an equivalent Application Load Balancer or Network Load Balancer. It reduces the time and effort required to migrate a Classic Load Balancer compared to other methods.

> **Note**
>
> The wizard creates a new load balancer. The wizard doesn't convert the existing Classic Load Balancer to an Application Load Balancer or Network Load Balancer. You must manually redirect the traffic to the newly created load balancer.

Limitations

- The name of the new load balancer can't be the same as an existing load balancer of the same type, in the same region.
- If the Classic Load Balancer has any tags containing the `aws:` prefix in their key, those tags are not migrated.

When migrating to an Application Load Balancer

- If the Classic Load Balancer has only one subnet, you must specify a second subnet.
- If the Classic Load Balancer has HTTP/HTTPS listeners that use TCP health checks, the health check protocol is updated to HTTP and the path is set to "/".
- If the Classic Load Balancer has HTTPS listeners using a custom or unsupported security policy, migration wizard uses the default security policy for the new load balancer type.

When migrating to a Network Load Balancer

- The following instance types will not be registered with the new target group: C1, CC1, CC2, CG1, CG2, CR1, CS1, G1, G2, HI1, HS1, M1, M2, M3, T1
- Certain health check settings from your Classic Load Balancer may not be transferrable to the new target group. These cases will be indicated as a change in the summary section of the migration wizard.
- If the Classic Load Balancer has SSL listeners, migration wizard creates a TLS listener using the certificate and security policy from the SSL listener.

Migration wizard process

To migrate a Classic Load Balancer using migration wizard

1. Open the Amazon EC2 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ec2/.
2. On the navigation pane, under **Load Balancing**, choose **Load Balancers**.
3. Select the Classic Load Balancer you want to migrate.
4. In the load balancers **Details** section, choose **Launch migration wizard**.
5. Choose **Migrate to Application Load Balancer**, or **Migrate to Network Load Balancer**, to open migration wizard.
6. Under **Name new load balancer**, for **Load balancer name** enter a name for your new load balancer.
7. Under **Name new target group and review targets**, for **Target group name** enter a name for your new target group.
8. (Optional) Under **Targets**, you can review the target instances that will be registered with the new target group.

9. (Optional) Under **Review tags**, you can review the tags that will be applied to your new load balancer
10. Under **Summary for Application Load Balancer**, or **Summary for Network Load Balancer**, review and verify the configuration options assigned by migration wizard.
11. After you're satisfied with the configuration summary, choose **Create Application Load Balancer**, or **Create Network Load Balancer**, to start the migration.

Migrate using the load balancer copy utility

The load balancer copy utilities are available within the ELB Tools repository, on the AWS GitHub page.

Resources

- ELB Tools
- Classic Load Balancer to Application Load Balancer copy utility
- Classic Load Balancer to Network Load Balancer copy utility

Migrate your load balancer manually

The following information provides general instructions for manually creating a new Application Load Balancer or Network Load Balancer based on an existing Classic Load Balancer in a VPC. You can migrate using the AWS Management Console, the AWS CLI, or an AWS SDK. For more information, see Getting started with ELB.

After you have completed the migration process, you can take advantage of the features of your new load balancer.

Manual migration process

Step 1: Create a new load balancer

Create a load balancer with a configuration that is equivalent to the Classic Load Balancer to migrate.

1. Create a new load balancer, with the same scheme (internet-facing or internal), subnets, and security groups as the Classic Load Balancer.

2. Create one target group for your load balancer, with the same health check settings that you have for your Classic Load Balancer.
3. Do one of the following:
   - If your Classic Load Balancer is attached to an Auto Scaling group, attach your target group to the Auto Scaling group. This also registers the Auto Scaling instances with the target group.
   - Register your EC2 instances with your target group.
4. Create one or more listeners, each with a default rule that forwards requests to the target group. If you create an HTTPS listener, you can specify the same certificate that you specified for your Classic Load Balancer. We recommend that you use the default security policy.
5. If your Classic Load Balancer has tags, review them and add the relevant tags to your new load balancer.

Step 2: Gradually redirect traffic to your new load balancer

After your instances are registered with your new load balancer, you can begin the process of redirecting traffic from the old load balancer to the new load balancer. This enables you to test your new load balancer while minimizing risk to the availability of your application.

To redirect traffic gradually to your new load balancer

1. Paste the DNS name of your new load balancer into the address field of an internet-connected web browser. If everything is working, the browser displays the default page of your application.
2. Create a new DNS record that associates your domain name with your new load balancer. If your DNS service supports weighting, specify a weight of 1 in the new DNS record and a weight of 9 in the existing DNS record for your old load balancer. This directs 10% of the traffic to the new load balancer and 90% of the traffic to the old load balancer.
3. Monitor your new load balancer to verify that it is receiving traffic and routing requests to your instances.

> **Important**
>
> The time-to-live (TTL) in the DNS record is 60 seconds. This means that any DNS server that resolves your domain name keeps the record information in its cache for 60 seconds, while the changes propagate. Therefore, these DNS servers can still route

traffic to your old load balancer for up to 60 seconds after you complete the previous step. During propagation, traffic could be directed to either load balancer.

4. Continue to update the weight of your DNS records until all traffic is directed to your new load balancer. When you are finished, you can delete the DNS record for your old load balancer.

Step 3: Update policies, scripts, and code

If you migrated your Classic Load Balancer to an Application Load Balancer or Network Load Balancer, be sure to do the following:

- Update IAM policies that use API version 2012-06-01 to use version 2015-12-01.
- Update processes that use CloudWatch metrics in the `AWS/ELB` namespace to use metrics from the `AWS/ApplicationELB` or `AWS/NetworkELB` namespace.
- Update scripts that use **aws elb** AWS CLI commands to use **aws elbv2** AWS CLI commands.
- Update CloudFormation templates that use the `AWS::ElasticLoadBalancing::LoadBalancer` resource to use the `AWS::ElasticLoadBalancingV2` resources.
- Update code that uses ELB API version 2012-06-01 to use version 2015-12-01.

Resources

- elbv2 in the *AWS CLI Command Reference*
- Elastic Load Balancing API Reference version 2015-12-01
- Identity and access management for ELB
- Application Load Balancer metrics in the *User Guide for Application Load Balancers*
- Network Load Balancer metrics in the *User Guide for Network Load Balancers*
- AWS::ElasticLoadBalancingV2::LoadBalancer in the *AWS CloudFormation User Guide*

Step 4: Delete the old load balancer

You can delete the old Classic Load Balancer after:

- You have redirected all traffic from the old load balancer to the new load balancer.
- All existing requests that were routed to the old load balancer have completed.

Prevent users from creating Classic Load Balancers

You can create an IAM policy that prevents users from creating Classic Load Balancers in your account.

Both the ELB V2 and ELB V1 APIs provide a `CreateLoadBalancer` API action. When you create a Classic Load Balancer, you use the V1 API action, which creates both the load balancer and listeners. When you create an Application Load Balancer, Network Load Balancer, or Gateway Load Balancer, you use the V2 API action, which creates only the load balancer. The V2 API provides a `CreateListener` action, which you use to create listeners for a load balancer after you create it.

The following policy denies users permission to create a load balancer if the listener protocol is specified. Because you must configure at least one listener when you create a Classic Load Balancer, this policy prevents users from creating Classic Load Balancers. It does not prevent users from creating other types of load balancers, because there are separate API actions for creating those load balancers and their listeners.

```
{
    "Version": "2012-10-17",
    "Effect": "Deny",
    "Action": "elasticloadbalancing:CreateLoadBalancer",
    "Resource": [
        "arn:aws:elasticloadbalancing:*:*:loadbalancer/*"
    ],
    "Condition": {
        "Null": {
            "elasticloadbalancing:ListenerProtocol": false
        }
    }
}
```