Amazon Kinesis Video Streams WebRTC Developer Guide

Kinesis Video Streams

Kinesis Video Streams: Amazon Kinesis Video Streams WebRTC Developer Guide

Table of Contents

What is Amazon Kinesis Video Streams with WebRTC?

WebRTC is an open technology specification for enabling real-time communication (RTC) across browsers and mobile applications via simple APIs. It uses peering techniques for real-time data exchange between connected peers and provides low latency media streaming required for human-to-human interaction. The WebRTC specification includes a set of IETF protocols including Interactive Connectivity Establishment, Traversal Using Relay around NAT (TURN), and Session Traversal Utilities for NAT (STUN) for establishing peer-to-peer connectivity, in addition to protocol specifications for reliable and secure real-time media and data streaming.

Amazon Kinesis Video Streams provides a standards-compliant WebRTC implementation as a fully managed capability. You can use Amazon Kinesis Video Streams with WebRTC to securely live stream media or perform two-way audio or video interaction between any camera IoT device and WebRTC-compliant mobile or web players. As a fully managed capability, you don't have to build, operate, or scale any WebRTC-related cloud infrastructure, such as signaling or media relay servers to securely stream media across applications and devices.

Using Kinesis Video Streams with WebRTC, you can easily build applications for live peer-to-peer media streaming, or real-time audio or video interactivity between camera IoT devices, web browsers, and mobile devices for a variety of use cases. Such applications can help parents keep an eye on their baby’s room, enable homeowners to use a video doorbell to check who’s at the door, enable owners of camera-enabled robot vacuums to remotely control the robot by viewing the live camera stream on a mobile phone, and so on.

If you're a first-time user of Kinesis Video Streams with WebRTC, we recommend that you read the following sections:

- How it works
- Amazon Kinesis Video Streams with WebRTC SDK in C for embedded devices
- Amazon Kinesis Video Streams with WebRTC SDK in JavaScript for web applications
- Amazon Kinesis Video Streams WebRTC SDK for Android
- Amazon Kinesis Video Streams WebRTC SDK for iOS
- Control plane APIs
- Data plane REST APIs
- Data plane Websocket APIs

Region availability

Note

Amazon Kinesis Video Streams with WebRTC is not yet supported in the AWS GovCloud (US) Region.

Amazon Kinesis Video Streams with WebRTC is available in the following regions:

Region Name	AWS Region Code
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Malaysia)	ap-southeast-5
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1

Region Name	AWS Region Code
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Spain)	eu-south-2
South America (Sao Paulo)	sa-east-1
Middle East (Bahrain)	me-south-1

How it works

Amazon Kinesis Video Streams with WebRTC enables real-time video communication between web browsers, mobile devices, and other WebRTC-enabled applications. It provides a secure and scalable infrastructure for building video streaming applications, handling tasks such as signaling, media streaming, and integration with other AWS services. This section explains the underlying architecture, components, and workflows of the service, helping explain its design principles and mechanisms.

Topics

Key concepts

The following are key terms and concepts specific to the Amazon Kinesis Video Streams with WebRTC.

Signaling channel

A resource that enables applications to discover, set up, control, and terminate a peer-to-peer connection by exchanging signaling messages. Signaling messages are metadata that two applications exchange with each other to establish peer-to-peer connectivity. This metadata includes local media information, such as media codecs and codec parameters, and possible network candidate paths for the two applications to connect with each other for live streaming.

Streaming applications can maintain persistent connectivity with a signaling channel and wait for other applications to connect to them. Or, they can connect to a signaling channel only when they need to live stream media. A signaling channel enables applications to connect with each other in a one-to-few model, using the concept of one master connecting to multiple viewers. The application that initiates the connection assumes the responsibility of a master using the `ConnectAsMaster` API and waits for viewers. Up to 10 applications can then connect to that signaling channel by assuming the viewer responsibility by invoking the `ConnectAsViewer` API. After they're connected to a signaling channel, the master and viewer applications can send each other signaling messages to establish peer-to-peer connectivity for live media streaming.

Peer

Any device or application (for example, a mobile or web application, web cam, home security camera, baby monitor, etc.) that is configured for real-time, two-way streaming through a Kinesis Video Streams with WebRTC.

Master

A peer that initiates the connection and is connected to the signaling channel with the ability to discover and exchange media with any of the signaling channel's connected viewers.

> **Important**
>
> Currently, a signaling channel can only have one master.

Viewer

A peer that is connected to the signaling channel with the ability to discover and exchange media only with the signaling channel's master. A viewer cannot discover or interact with other viewers through a given signaling channel. A signaling channel can have up to 10 connected viewers.

Technology concepts

As you get started with Kinesis Video Streams with WebRTC, you can also benefit from learning about several interrelated protocols and APIs of which the WebRTC technology consists.

Session Traversal Utilities for NAT (STUN)

A protocol that is used to discover your public address and determine any restrictions in your router that would prevent a direct connection with a peer.

Traversal Using Relays around NAT (TURN)

A server that is used to bypass the Symmetric NAT restriction by opening a connection with a TURN server and relaying all information through that server.

Session Description Protocol (SDP)

A standard for describing the multimedia content of the connection such as resolution, formats, codecs, encryption, etc. so that both peers can understand each other once the data is transferring.

SDP Offer

An SDP message sent by an agent which generates a session description in order to create or modify a session. It describes the aspects of desired media communication.

SDP Answer

An SDP message sent by an answerer in response to an offer received from an offerer. The answer indicates the aspects that are accepted. For example, if all the audio and video streams in the offer are accepted.

Interactive Connectivity Establishment (ICE)

A framework that allows your web browser to connect with peers.

ICE Candidate

A method that the sending peer is able to use to communicate.

How STUN, TURN and ICE work together

Let's take the scenario of two peers, A and B, who are both using a WebRTC peer to peer two way media streaming (for example, a video chat application). What happens when A wants to call B?

To connect to B's application, A's application must generate an SDP offer. An SDP offer contains information about the session A's application wants to establish, including what codecs to use, whether this is an audio or video session, etc. It also contains a list of ICE candidates, which are the IP and port pairs that B's application can attempt to use to connect to A.

To build the list of ICE candidates, A's application makes a series of requests to a STUN server. The server returns the public IP address and port pair that originated the request. A's application adds each pair to the list of ICE candidates, in other words, it gathers ICE candidates. Once A's application has finished gathering ICE candidates, it can return an SDP.

Next, A's application must pass the SDP to B's application through a signaling channel over which these applications communicate. The transport protocol for this exchange is not specified in the WebRTC standard. It can be performed over HTTPS, secure WebSocket, or any other communication protocol.

Now, B's application must generate an SDP answer. B's application follows the same steps A used in the previous step: gathers ICE candidates, etc. B's application then needs to return this SDP answer to A's application.

After A and B have exchanged SDPs, they then perform a series of connectivity checks. The ICE algorithm in each application takes a candidate IP/port pair from the list it received in the other party's SDP, and sends it a STUN request. If a response comes back from the other application, the originating application considers the check successful and marks that IP/port pair as a valid ICE candidate.

After connectivity checks are finished on all of the IP/port pairs, the applications negotiate and decide to use one of the remaining, valid pairs. When a pair is selected, media begins flowing between the application.

If either of the applications can't find an IP/port pair that passes connectivity checks, they'll make STUN requests to the TURN server to obtain a media relay address. A relay address is a public IP address and port that forwards packets received to and from the application to set up the relay address. This relay address is then added to the candidate list and exchanged via the signaling channel.

Components

Kinesis Video Streams with WebRTC includes the following components:

- **Control plane**

The control plane component is responsible for creating and maintaining the Kinesis Video Streams with WebRTC signaling channels. For more information, see the Amazon Kinesis Video Streams API Reference.

- **Signaling**

  The signaling component manages the WebRTC signaling endpoints that allow applications to securely connect with each other for peer-to-peer live media streaming. The signaling component includes the Amazon Kinesis Video Signaling REST APIs and a set of Websocket APIs.
- **STUN**

  This component manages STUN endpoints that enable applications to discover their public IP address when they are located behind a NAT or a firewall.
- **TURN**

  This component manages TURN endpoints that enable media relay via the cloud when applications can't stream media peer-to-peer.
- **Kinesis Video Streams WebRTC SDKs**

  These are software libraries that you can download, install, and configure on your devices and application clients to enable your camera IoT devices with WebRTC capabilities to engage in low latency peer-to-peer media streaming. These SDKs also enable Android, iOS, and web application clients to integrate Kinesis Video Streams with WebRTC signaling, TURN, and STUN capabilities with any WebRTC-compliant mobile or web players.
  - Amazon Kinesis Video Streams with WebRTC SDK in C for embedded devices
  - Amazon Kinesis Video Streams with WebRTC SDK in JavaScript for web applications
  - Amazon Kinesis Video Streams WebRTC SDK for Android
  - Amazon Kinesis Video Streams WebRTC SDK for iOS

System requirements

This section covers basic system requirements for using Amazon Kinesis Video Streams with WebRTC, including network requirements and environment. It also includes information about debugging connections.

Networking requirements

The general networking requirements for the signaling channel service endpoints for Kinesis Video Streams with WebRTC are:

- HTTPS calls to endpoints hosted at `https://*.kinesisvideo.{region}.amazonaws.com`
- WebSocket integrations with endpoints `wss://*.kinesisvideo.{region}.amazonaws.com`
- STUN servers at `stun:stun.kinesisvideo.{aws-region}.amazonaws.com:443`
- TURN servers at `turn:_._.kinesisvideo.{aws-region}.amazonaws.com:443` and `turns:_._.kinesisvideo.{aws-region}.amazonaws.com:443`

Note

IPv6 addresses aren't currently supported for STUN and TURN servers.

The protocols used between peers as part of the RTCPeerConnection can be either TCP or UDP based.

Most applications attempt to establish direct, peer-to-peer connections by determining the IP addresses for each peer, as well as the ports and protocols to be exchanged as ICE candidates. These candidates are used to attempt to connect to one another using these candidates. They will attempt each pair until a connection can be established.

Network environment

If a message from the viewer has been sent to the master and a log such as `No valid ICE candidate` is recorded, then no valid connection route has been found. This can happen if there is a firewall that prevents direct connection, or if the network is not reachable.

Do the following to troubleshoot the connectivity issue:

- If you're not using TURN on the master side, be sure to enable TURN.

  TURN is enabled by default in the C SDK. In the JavaScript SDK, select STUN/TURN or `TURN only` in NAT Traversal.
- For restricted networks, such as an enterprise network, try other available networks (wired or wireless).

If you're connecting to a VPN, disconnect from it.

Note

You can disregard 403 `Forbidden IP` errors returned by Kinesis Video Streams TURN servers. The servers reject ICE candidate pairs that contain `localhost` IPs, such as `192.168.*` or `10.0.0.*`.
This may cause some, but not all, ICE candidate pairs to fail.

Debugging ongoing connections

There are a few areas that can cause issues with an established, ongoing WebRTC connection, but networking is the most common.

You can confirm a VERBOSE level log from the SDK by setting `export AWS_KVS_LOG_LEVEL=1` as an environment variable.

Note

If no candidate pair is found within the allotted timeout, the ICE agent returns error status `0x5a00000d`.

For additional information about log levels, see GitHub.

```
export AWS_KVS_LOG_LEVEL=1
./kvsWebrtcClientMasterGstSample TestChannel
```

You will see logs like the following. From these logs, you can confirm the Interactive Connectivity Establishment (ICE) candidate (IP address and port) and the selected candidate pair.

```
2023-02-13 05:57:16 DEBUG   iceAgentReadyStateSetup(): Selected pair w1UdV9fE+_/
CuBel1pl, local candidate type: srflx. Round trip time 7 ms. Local candidate priority:
 1694498815, ice candidate pair priority: 7240977859836116990
2023-02-13 05:57:16 INFO    onConnectionStateChange(): New connection state 3
2023-02-13 05:57:16 DEBUG   rtcPeerConnectionGetMetrics(): ICE local candidate Stats
 requested at 16762678365731494
```

```
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Local Candidate IP
 Address: XXX.XXX.XXX.XXX
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Local Candidate
 type: srflx
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Local Candidate
 port: 38563
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Local Candidate
 priority: 1694498815
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Local Candidate
 transport protocol: udp
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Local Candidate
 relay protocol: N/A
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Local Candidate Ice
 server source: stun.kinesisvideo.ap-northeast-1.amazonaws.com
2023-02-13 05:57:16 DEBUG   rtcPeerConnectionGetMetrics(): ICE remote candidate Stats
 requested at 16762678365732111
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Remote Candidate IP
 Address: XXX.XXX.XXX.XXX
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Remote Candidate
 type: srflx
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Remote Candidate
 port: 45867
2023-02-13 05:57:16 VERBOSE signalingClientGetCurrentState(): Signaling Client Get
 Current State
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Remote Candidate
 priority: 1685921535
2023-02-13 05:57:16 DEBUG   logSelectedIceCandidatesInformation(): Remote Candidate
 transport protocol: udp
```

Amazon Kinesis Video Streams with WebRTC service quotas

Kinesis Video Streams with WebRTC has the following service quotas:

Important

The following service quotas are either soft **[s]**, which can be increased, or hard **[h]**, which can't be increased. You will see [s] and [h] next to individual service quota in the tables below.

Note

TPS stands for *transactions per second*.

Topics

- Control plane API service quotas
- Signaling API service quotas
- TURN service quotas
- WebRTC ingestion service quotas
- Troubleshooting

Control plane API service quotas

The following section describes service quotas for the control plane APIs.

API	Maximum API request rate per AWS account	Maximum number of channels per AWS account per AWS Region	Maximum API request per channel
CreateSignalingChannel	50 TPS [s]	• For US East (N. Virginia) (us-east-1) and US West (Oregon) (us-west-2) only - 10,000 • All other supported Regions - 5,000	N/A
DeleteSignalingChannel	50 TPS [h]	N/A	5 TPS [h]
DescribeMediaStorageConfiguration	50 TPS [h]	N/A	5 TPS [h]

API	Maximum API request rate per AWS account	Maximum number of channels per AWS account per AWS Region	Maximum API request per channel
DescribeSignalingChannel	300 TPS [h]	N/A	5 TPS [h]
GetSignalingChannelEndpoint	300 TPS [h]	N/A	N/A
ListSignalingChannels	50 TPS [h]	N/A	N/A
ListTagsForResource	50 TPS [h]	N/A	5 TPS [h]
TagResource	50 TPS [h]	N/A	5 TPS [h]
UntagResource	50 TPS [h]	N/A	5 TPS [h]
UpdateMediaStorageConfiguration	10 TPS [h]	N/A	5 TPS [h]
UpdateSignalingChannel	50 TPS [h]	N/A	5 TPS [h]

Signaling API service quotas

The following section describes service quotas for the signaling component in Kinesis Video Streams with WebRTC. For more information, see How it works.

API	Maximum GO_AWAY message grace period prior to terminating connection	Maximum API request rate per channel	Maximum number of concurrent connections per channel	Maximum connection duration	Maximum idle connection timeout period
ConnectAsMaster	1 minute (h)	3 TPS (h)	1 (h)	1 hour (h)	10 minutes (h)
ConnectAsViewer	1 minute (h)	3 TPS (h)	10 (s)	1 hour (h)	10 minutes (h)

API	Maximum API request rate	Maximum message payload size
SendAlexaOffertoMaster	• 5 TPS per signaling channel (h) • 100 TPS per AWS Region per AWS account (s)	N/A
SendICECandidate	20 TPS per WebSocket connection (h)	10k (h)
SendSDPAnswer	5 TPS per WebSocket connection (h)	10k (h)
SendSDPOffer	5 TPS per WebSocket connection (h)	10k (h)

TURN service quotas

The following section describes service quotas for the Traversal Using Relays around NAT (TURN) component in Kinesis Video Streams with WebRTC. For more information, see How it works.

API or parameter	Value
GetIceServerConfig	• 5 TPS per signaling channel (h) • 100 TPS per AWS Region per AWS account (s)
Bit Rate	5Mbps (h)
Credential Lifecycle	5 minutes (h)
Number of allocations	50 per signaling channel (h)

WebRTC ingestion service quotas

The following section describes service quotas for the media recording component in Amazon Kinesis Video Streams WebRTC. For more information, see Use Amazon Kinesis Video Streams with WebRTC to ingest and store media.

JoinStorageSession

- API:
  - Per AWS account - 50 TPS (h)
  - Per channel - 2 TPS (h)
- Streaming session quotas:
  - Bit rate - 1 Mbps (s)
  - Session duration - 1 hour (h)
  - Idle timeout - 3 minutes (h)

JoinStorageSessionAsViewer

- API:
  - Per AWS account - 50 TPS (h)
  - Per channel - 2 TPS (h)
- Streaming session quotas:
  - Max concurrent clients in a session - 3 count (h)

- Session duration - 1 hour (h)
- Idle timeout - 1 minutes (h)

Troubleshooting

You can only connect **one** master and **one or more** viewers to a single signaling channel.

It isn't possible to connect multiple masters to a single signaling channel.

Access Kinesis Video Streams with WebRTC

You can work with Kinesis Video Streams with WebRTC in any of the following ways:

AWS Management Console

Getting Started with the AWS Management Console

The console is a browser-based interface to access and use AWS services, including Kinesis Video Streams with WebRTC.

AWS SDKs

AWS provides software development kits (SDKs) that consist of libraries and sample code for various programming languages and platforms (for example, Java, Python, Ruby, .NET, iOS, Android, and more). The SDKs provide a convenient way to create programmatic access to Kinesis Video Streams with WebRTC. For information about the AWS SDKs, including how to download and install them, see Tools for Amazon Web Services.

Kinesis Video Streams with WebRTC HTTPS API

You can access Kinesis Video Streams with WebRTC and AWS programmatically by using the Kinesis Video Streams with WebRTC APIs, which lets you issue API requests directly to the service. For more information, see the Amazon Kinesis Video Streams API reference.

Getting started

This section describes how to perform the following tasks in Amazon Kinesis Video Streams with WebRTC:

- Set up your AWS account and create an administrator.
- Create a Kinesis Video Streams with WebRTC signaling channel.
- Set up Kinesis Video Streams with WebRTC on Ingenic T31 hardware.

If you are new to Kinesis Video Streams with WebRTC, we recommend that you read How it works first.

Set up an AWS account

Before you use Kinesis Video Streams with WebRTC for the first time, complete the following tasks:

Topics

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.eu/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Create an AWS account key

You need an AWS account key to access Kinesis Video Streams with WebRTC programmatically.

To create an AWS account key, do the following:

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. Choose **Users** in the navigation bar, and choose the **Administrator** user.
3. Choose the **Security credentials** tab, and choose **Create access key**.
4. Record the **Access key ID**. Choose **Show** under **Secret access key**, and then record the **Secret access key**.

Create a signaling channel

A Kinesis Video Streams with WebRTC signaling channel facilitates the exchange of signaling messages required to establish and maintain peer-to-peer connections between WebRTC clients. It handles the negotiation of Session Description Protocol (SDP) offers and answers for session parameters, as well as the exchange of Interactive Connectivity Establishment (ICE) candidates for network information.

To create a signaling channel, call the CreateSignalingChannel API. This page will show you how to invoke that API using the AWS Management Console, AWS CLI, and one of the AWS SDKs.

Important

Make note of the channel ARN, you'll need it later.

AWS Management Console

Do the following:

1. Open the **Kinesis Video Streams Signaling Channels** console at https://console.aws.amazon.com/kinesisvideo/home/#/signalingChannels.
2. Choose **Create signaling channel**.
3. On the **Create a new signaling channel** page, type the name for the signaling channel.

   Leave the default **Time-to-live (Ttl)** value as 60 seconds.

   Choose **Create signaling channel**.
4. Once the signaling channel is created, review the details on the channel's details page.

AWS CLI

Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface User Guide.

For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Run the following Create-Signaling-Channel command using the AWS CLI:

```
aws kinesisvideo create-signaling-channel \
  --channel-name "YourChannelName" \
  --region "us-west-2"
```

The response will look like the following:

```
{
    "ChannelARN": "arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123"
}
```

AWS SDK

This code snippet shows you how to create a Kinesis Video Streams with WebRTC signaling channel using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same. View a complete code example on GitHub.

Create the Kinesis Video Streams client. This is the client used to call the `CreateSignalingChannel` API.

```
const clientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2'
};
const kinesisVideoClient = new AWS.KinesisVideo(clientConfig);
```

Use the client to call the `CreateSignalingChannel` API.

```
const createSignalingChannelResponse = await kinesisVideoClient
    .createSignalingChannel({
        ChannelName: 'YourChannelName',
    })
    .promise();
```

Print the response.

```
console.log(createSignalingChannelResponse.ChannelARN);
```

The live web page with this code sample is available for use on GitHub. Input your region, AWS credentials, and the name of your signaling channel.

Select **Create Channel**.

Integrate with Ingenic T31

Note

AWS does not endorse this chipset in any way, nor do we guarantee the integration will work. This guide is based on testing by the Amazon Kinesis Video Streams team and is provided to assist customers in setting up their devices.

Follow these procedures to set up Amazon Kinesis Video Streams with WebRTC on Ingenic T31 hardware.

Download the code

1. Set up a build directory.

   For this tutorial, create a directory called `ingenic` within the `Downloads` folder.

   ```
   mkdir ~/Downloads/ingenic
   cd ~/Downloads/ingenic
   ```

2. Clone the following repo into your new directory: https://github.com/aws-samples/amazon-kinesis-video-streams-media-interface.

   ```
   git clone https://github.com/aws-samples/amazon-kinesis-video-streams-media-
   interface.git
   ```

Set up the build environment

1. Obtain the pre-built docker image containing the tool chain and place it in the `ingenic` folder.

   Important

   The tool chain may differ based on the board and CPU types. Check with the vendor for the correct tool chain.

2. Load the `Ingenic-T31-app-build.tar.gz` docker image.

```
docker load --input ./Ingenic-T31-app-build.tar.gz
```

Expected output:

```
580272b5675c: Loading layer [==================================================>]
 1.666GB/1.666GB
Loaded image ID:
 sha256:76d41ef9b2f53ad3f2a33f00ae110df3d1b491378a4005e19ea989ce97e99bc1
```

Make note of the `Image Id`. You'll need this in the next step.

3. Run the following command to mount the current directory (`~/Downloads/ingenic`) on your host machine to the `/ingenicMappedFolder` directory inside the Docker container.

   This command also runs the container in interactive mode with a terminal session. You'll be dropped into a Bash shell inside the container, which allows you to execute commands directly.

```
docker run \
   -v `pwd`:/ingenicMappedFolder \
   -it 76d41ef9b2f53ad3f2a33f00ae110df3d1b491378a4005e19ea989ce97e99bc1 \
   /bin/bash
```

4. Review the contents of the `/ingenicMappedFolder` folder to confirm that the bind-mount volume succeeded.

   Prompt:

```
ls /ingenicMappedFolder
```

   Response:

```
AWS-Solution-Remote-Diagnostic-Media-Interface Ingenic-T31-app-build.tar.gz
```

5. Verify that the docker container has the environment variables set up:

   Prompt:

```
env
```

Response:

```
CC=mips-linux-gnu-gcc
CXX=mips-linux-gnu-g++
PATH=/mips-gcc540-glibc222-64bit-r3.3.0/bin:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin
```

6. Verify that the docker container contains the tool chain and Ingenic SDK. It should be located at `/ingenic-sdk`.

   Prompt:

   ```
   ls /
   ```

   Response:

   ```
   bin   ingenic-sdk          libx32                             proc  sys
   boot  ingenicMappedFolder  media                              root  tmp
   dev   lib                  mips-gcc540-glibc222-64bit-r3.3.0  run   usr
   etc   lib32                mnt                                sbin  var
   home  lib64                opt                                srv
   ```

Build the Amazon Kinesis Video Streams with WebRTC application

1. Type the following:

   ```
   cd /ingenicMappedFolder/AWS-Solution-Remote-Diagnostic-Media-Interface
   cp -r /ingenic-sdk/* \
       /ingenicMappedFolder/amazon-kinesis-video-streams-media-interface/3rdparty/T31/
   export LDFLAGS=-Wl,--dynamic-linker=/lib/ld.so.1
   mkdir build
   cmake -B ./build -DBOARD=T31 -DCMAKE_BUILD_TYPE=Release -DBUILD_WEBRTC_SAMPLES=ON \
       -DBUILD_KVS_SAMPLES=ON -DBUILD_SAVE_FRAME_SAMPLES=ON
   cmake --build ./build --config Release
   ```

   The `kvswebrtcmaster-static` application is located at `/ingenicMappedFolder/AWS-Solution-Remote-Diagnostic-Media-Interface/build/samples/webrtc/`.

2. Run the following command to exit the docker container:

```
exit
```

3. On your host machine, verify that the `kvswebrtcmaster-static` application is present.

```
ls ~/Downloads/ingenic/AWS-Solution-Remote-Diagnostic-Media-Interface/build/
samples/webrtc/kvswebrtcmaster-static
```

Upload the application to the device

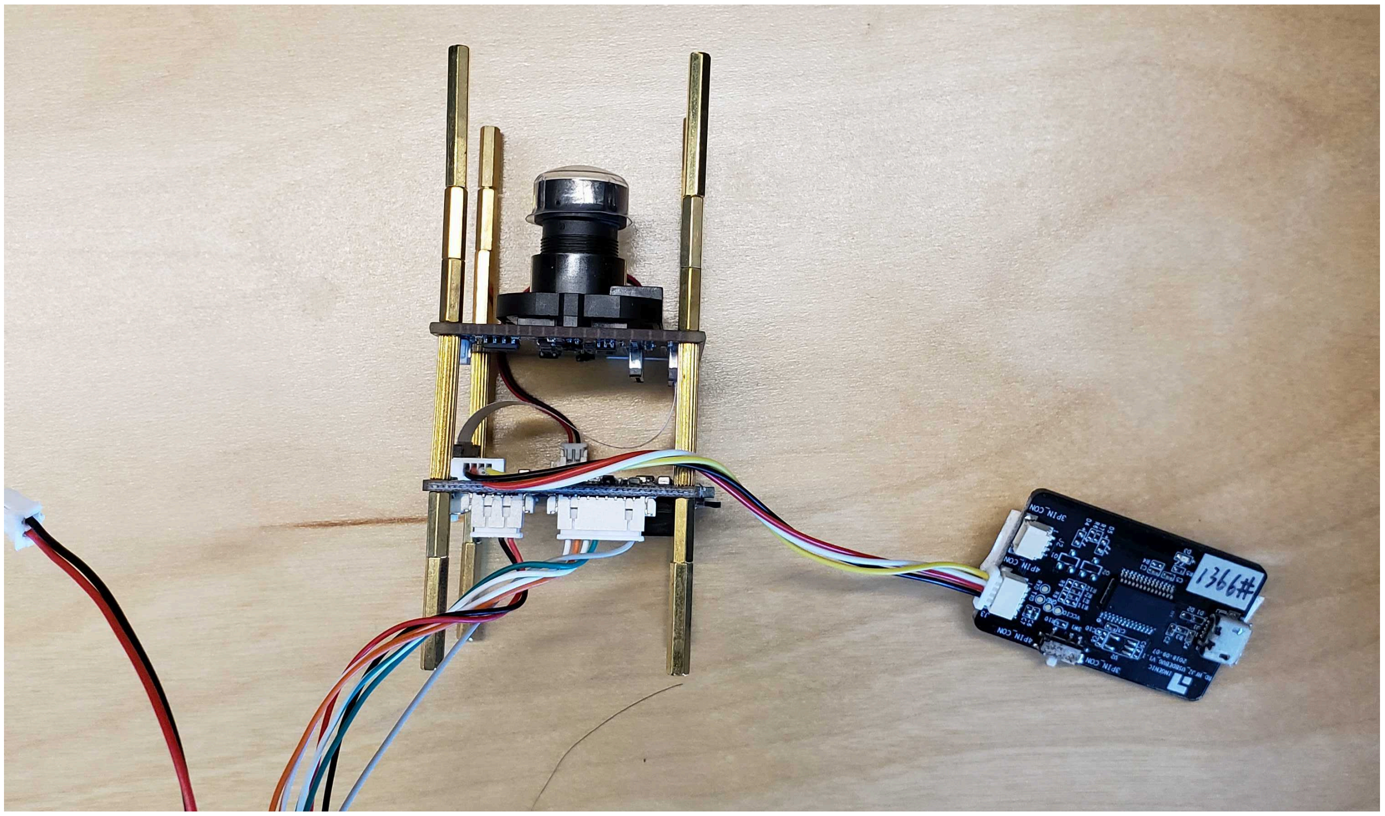
1. On your host machine, copy the static binary to a micro SD Card.

```
cp ~/Downloads/ingenic/AWS-Solution-Remote-Diagnostic-Media-Interface/build/
samples/webrtc/kvswebrtcmaster-static /Volumes/IngenicSDCard
```

2. Download this .pem file and place it in the micro SD card as `cert.pem`.
3. Put the micro SD card into the micro SD card slot.

Connect to the device

1. Attach the serial port tool to the board, as shown below.



2. Plug the ethernet and power cords into the device. A red LED should turn on.
3. Connect your host machine to the micro-usb slot on the serial port tool.
4. Check for the connected devices:

```
ls /dev/tty.*
```

> **Note**
>
> If you have multiple TTY devices, determine which TTY device is the Ingenic board. Disconnect the device from your host machine and run `ls` again. Compare the command output and locate the difference.

5. Connect to it using `screen` or another serial port tool (for example, Tera Term). Set the baud rate to 115200.

```
screen /dev/tty.usbserial-XXXXXXX 115200
```

6. Determine the appropriate action, based on the state of your board:

- If the shell session ends with a `#`, the boot was interrupted. Use `boot` command to start the Linux OS, then continue with the rest of this step.
- If the shell session asks you to log in, type your password.
- If the screen is blank, press Enter.
- If the screen shows the error `Cannot exec '/dev/tty.usbserial-XXXXXXX': No such file or directory`, double-check all of the physical connections between the board and your host machine.

Mount the micro SD card on the board

1. Create a directory:

   Note

   For this example, we're using `sdcard`.

   ```
   mkdir /tmp/sdcard
   ```

2. Type the following to mount the micro SD card to the folder:

   ```
   mount /dev/mmcblk0p1 /tmp/sdcard
   ```

3. Review the contents of the folder:

   ```
   ls /tmp/sdcard
   ```

Run the application

1. Navigate to the mounted directory:

   ```
   cd /tmp/sdcard
   ```

2. Export your credentials and other information from the board:

   ```
   export AWS_ACCESS_KEY_ID=ID
   ```

```
export AWS_SECRET_ACCESS_KEY=key
export AWS_DEFAULT_REGION=us-west-2
export AWS_KVS_CACERT_PATH=`pwd`/cert.pem
```

3. Run the application:

```
./kvswebrtcmaster-static channel-name
```

View the media

To view the media, connect to the signaling channel as a **viewer**. See the following sections for examples:

- JavaScript
- AWS Management Console
- Android
- iOS

Troubleshooting

This section contains common questions and issues that we have encountered.

When I connect the board to my host machine and run `ls /dev/tty.*`, the device doesn't appear.

Verify the connections and all wires are secured, and that the device is powered on. Check that any LED indicators on your USB-to-TTY device are lit. If you're still having issues, contact the vendor to further diagnose issues connecting to the board.

The application fails to connect to signaling

If receive one of the following errors:

```
SSL error: certificate is not yet valid
```

or

```
2024-09-19 08:56:34.920 WARN    lwsHttpCallbackRoutine(): Received client http read
 response:  { "message": "Signature expired: 20240919T085634Z is now earlier than
 20240919T155135Z (20240919T155635Z - 5 min.)" }
```

It means the time is not correct on your Ingenic board. Verify this by running the date command. Set the time according to the vendor’s instructions, then run the application again.

The application fails to start, even though the file is there

If the error looks like:

```
-sh ./kvswebrtcmaster-static: not found
```

It means that the operating system wasn't able to find a dependency listed in the application's ELF header. For example, `libc` on the device.

Usually, this indicates that the toolchain used is not compatible with the board, or that the linker flags (`LDFLAGS`) don't correctly point to the required library paths, leading to unresolved dependencies during runtime.

Make sure that the correct toolchain (`uclibc` vs `glibc` and its version) matches the software on the board and set `LDFLAGS` according to the section called “Build with the WebRTC application”.

How do I unmount the micro SD card?

To unmount the micro SD card, use the command:

```
umount /tmp/sdcard
```

Stream live media (SDKs)

In Amazon Kinesis Video Streams WebRTC, peers are devices that are configured for real-time, two-way streaming via a signaling channel. Amazon Kinesis Video Streams with WebRTC SDKs are easy-to-use software libraries that you can download and install on the devices and application clients that you want to configure as peers over a given signaling channel.

Amazon Kinesis Video Streams with WebRTC includes the following SDKs:

- Amazon Kinesis Video Streams with WebRTC SDK in C for embedded devices
- Amazon Kinesis Video Streams with WebRTC SDK in JavaScript for web applications
- Amazon Kinesis Video Streams WebRTC SDK for Android
- Amazon Kinesis Video Streams WebRTC SDK for iOS

Each SDK includes corresponding samples and step-by-step instructions that can help you build and run those applications. You can use these samples for low latency, live, two-way audio and video streaming and data exchange between any combinations of Web/Android/iOS applications or embedded devices. In other words, you can stream live audio and video from an embedded camera device to Android or web applications or between two Android applications.

Amazon Kinesis Video Streams with WebRTC SDK in C for embedded devices

The following step-by-step instructions describe how to download, build, and run the Kinesis Video Streams with WebRTC SDK in C for embedded devices and its corresponding samples.

The following codecs are supported:

- **Audio:**
  - G.711 A-Law
  - G.711 U-Law
  - Opus
- **Video:**
  - H.264
  - H.265

- VP8

Download the SDK

To download the Kinesis Video Streams with WebRTC SDK in C for embedded devices, run the following command:

```
$ git clone https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-c.git
```

Build the SDK

Important

Before you complete these steps on a macOS and depending on the version of the macOS you have, you must run `xcode-select --install` to download the package with the command line tools and header. Then open `/Library/Developer/CommandLineTools/Packages/macOS_SDK_headers_for_macOS_10.14.pkg` and follow the installer to install the command line tools and header. You only need to do this once and before invoking `cmake`. If you already have the command line tools and header installed, you do not need to run this command again.

Complete the following steps:

1. Install cmake:
   - On macOS, run `brew install cmake pkg-config srtp`
   - on Ubuntu, run `sudo apt-get install pkg-config cmake libcap2 libcap-dev`
2. Obtain the access key and the secret key of the AWS account that you want to use for this demo.
3. Run the following command to create a `build` directory in your downloaded WebRTC C SDK, and execute `cmake` from it:

   ```
   $ mkdir -p amazon-kinesis-video-streams-webrtc-sdk-c/build; cd amazon-kinesis-video-streams-webrtc-sdk-c/build; cmake ..
   ```

4. Now that you're in the `build` directory you just created with the step above, run `make` to build the WebRTC C SDK and its provided samples.

> **Note**
>
> The `kvsWebrtcClientMasterGstSample` will NOT be built if the system doesn't have `gstreamer` installed. To make sure it is built (on macOS) you must run: `brew install gstreamer gst-plugins-base gst-plugins-good`

Run the SDK samples

After you complete the procedure above, you end up with the following sample applications in your `build` directory:

- `kvsWebrtcClientMaster` - This application sends sample H264/Opus frames (path: /samples/h264SampleFrames and /samples/opusSampleFrames) via the signaling channel. It also accepts incoming audio, if enabled in the browser. When checked in the browser, it prints the metadata of the received audio packets in your terminal.
- `kvsWebrtcClientViewer` - This application accepts sample H264/Opus frames and prints them out.
- `kvsWebrtcClientMasterGstSample` - This application sends sample H264/Opus frames from a GStreamer pipeline.

To run any of these samples, complete the following steps:

1. Setup your environment with your AWS account credentials:

```
export AWS_ACCESS_KEY_ID=YourAccessKey
export AWS_SECRET_ACCESS_KEY=YourSecretKey
export AWS_DEFAULT_REGION=YourAWSRegion
```

   If you're using temporary AWS credentials, also export your session token:

```
export AWS_SESSION_TOKEN=YourSessionToken
```

   If you have a custom CA certificate path to set, you can set it using:

```
export AWS_KVS_CACERT_PATH=../certs/cert.pem
```

> **Note**
>
> By default, the SSL CA certificate is set to ../certs/cert.pem which points to the file in this repository in GitHub.

2. Run either of the sample applications by passing to it the name that you want to give to your signaling channel. The application creates the signaling channel using the name you provide. For example, to create a signaling channel called `myChannel` and to start sending sample H264/Opus frames via this channel, run the following command:

   ```
   ./kvsWebrtcClientMaster myChannel
   ```

   When the command line application prints `Connection established`, you can proceed to the next step.

3. Now that your signaling channel is created and the connected master is streaming media to it, you can view this stream. For example, you can view this live stream in a web application. To do so, open the WebRTC SDK Test Page using the steps in Use the sample application and set the following values using the same AWS credentials and the same signaling channel that you specified for the master above:

   - Access key ID
   - Secret access key
   - Signaling channel name
   - Client ID (optional)

   Choose **Start viewer** to start live video streaming of the sample H264/Opus frames.

Video tutorial

This video demonstrates how to connect your camera and get started with Amazon Kinesis Video Streams for WebRTC.

Amazon Kinesis Video Streams with WebRTC SDK in JavaScript for web applications

You can find the Kinesis Video Streams with WebRTC SDK in JavaScript for web applications and its corresponding samples in GitHub.

Topics

Install the SDK

Whether and how you install the Kinesis Video Streams with WebRTC SDK in JavaScript depends on whether the code executes in `Node.js` modules or browser scripts.

NodeJS module

The preferred way to install the Kinesis Video Streams with WebRTC SDK in JavaScript for Node.js is to use npm, the Node.js package manager.

The package is hosted at https://www.npmjs.com/package/amazon-kinesis-video-streams-webrtc.

To install this SDK in your `Node.js` project, use the terminal to navigate to the same directory as your project's `package.json`:

Type the following:

```
npm install amazon-kinesis-video-streams-webrtc
```

You can import the SDK classes like typical Node.js modules:

```
// JavaScript
const SignalingClient = require('amazon-kinesis-video-streams-
webrtc').SignalingClient;
// TypeScript
```

```
import { SignalingClient } from 'amazon-kinesis-video-streams-webrtc';
```

Browser

You don't have to install the SDK to use it in browser scripts. You can load the hosted SDK package directly from AWS with a script in your HTML pages.

To use the SDK in the browser, add the following script element to your HTML pages:

```
<script src="https://unpkg.com/amazon-kinesis-video-streams-webrtc/dist/kvs-
webrtc.min.js"></script>
```

After the SDK loads in your page, the SDK is available from the global variable `KVSWebRTC` (or `window.KVSWebRTC`).

For example, `window.KVSWebRTC.SignalingClient`.

WebRTC JavaScript SDK documentation

The documentation for the SDK methods are on the GitHub readme, under Documentation.

In the Usage section, there is additional information for integrating this SDK along with the AWS SDK for JavaScript to build a web-based viewer application.

See the `examples` directory for an example of a complete application, including both a master and viewer role.

Use the sample application

Kinesis Video Streams with WebRTC also hosts a sample application that you can use to either create a new signaling channel or connect to an existing channel and use it as a master or viewer.

The Kinesis Video Streams with WebRTC sample application is located in GitHub.

The code for the sample application is in the `examples` directory.

Topics

- Stream peer-to-peer from the sample application to the AWS Management Console
- Stream peer-to-peer from the sample application to the sample application
- Stream peer-to-peer with WebRTC Ingestion from the sample page to the sample page

Stream peer-to-peer from the sample application to the AWS Management Console

1. Open the Kinesis Video Streams with WebRTC sample application and complete the following:

   - AWS Region. For example, `us-west-2`.
   - The AWS access key and the secret key for your IAM user or role. Leave the session token blank if you are using long-term AWS credentials.
   - The name of the signaling channel to which you want to connect.

     If you want to connect to a new signaling channel, choose **Create Channel** to create a signaling channel with the value provided in the box.

     Note

     Your signaling channel name must be unique for the current account and region. You can use letters, numbers, underscores (_), and hyphens (-), but not spaces.

   - Whether you want to send audio, video, or both.
   - WebRTC Ingestion and Storage. Expand the node and choose one of the following:
     - Select **Automatically determine ingestion mode**.
     - Make sure **Automatically determine ingestion mode** isn't selected and set the manual override to **OFF**.

     Note

     Automatically determine ingestion mode has the application call the DescribeMediaStorageConfiguration API to determine which mode to run in (Peer-to-peer or WebRTC ingestion). This additional API call adds a small amount to the startup time.
     If you know ahead of time which mode this signaling channel is running in, use the manual override to skip this API call.

   - ICE candidate generation. Leave STUN/TURN selected and leave `Trickle ICE` enabled.
2. Choose **Start Master** to connect to the signaling channel.

   Allow access to your camera and/or microphone, if needed.

3. Open the Kinesis Video Streams console in the AWS Management Console.

   Make sure the correct region is selected.

4. In the left navigation, select **signaling channels**.

   Select the name of the signaling channel above. Use the search bar, if needed.

5. Expand the **Media playback viewer** section.

6. Choose the **play** button on the video player. This joins the WebRTC session as a `viewer`.

   The media that is being sent on the demo page should display in the AWS Management Console.

Stream peer-to-peer from the sample application to the sample application

1. Open the Kinesis Video Streams with WebRTC sample application and complete the following information:

   - AWS Region. For example, `us-west-2`.
   - The AWS access key and the secret key for your IAM user or role. Leave the session token blank if you are using long-term AWS credentials.
   - The name of the signaling channel to which you want to connect.

     If you want to connect to a new signaling channel, choose **Create Channel** to create a signaling channel with the value provided in the box.

     Note

     Your signaling channel name must be unique for the current account and region. You can use letters, numbers, underscores (_), and hyphens (-), but not spaces.

   - Whether you want to send audio, video, or both.
   - WebRTC Ingestion and Storage. Expand the node and choose one of the following:
     - Select **Automatically determine ingestion mode**.
     - Make sure **Automatically determine ingestion mode** isn't selected and set the manual override to **OFF**.

> **Note**
>
> **Automatically determine ingestion mode** has the application call the DescribeMediaStorageConfiguration API to determine which mode to run in (Peer-to-peer or WebRTC ingestion). This additional API call adds a small amount to the startup time.
> If you know ahead of time which mode this signaling channel is running in, use the manual override to skip this API call.

   - ICE candidate generation. Leave `STUN/TURN` selected and leave `Trickle ICE` enabled.

2. Choose **Start Master** to connect to the signaling channel as the `master` role.

   Allow access to your camera and/or microphone, if needed.

3. Open another browser tab and open the Kinesis Video Streams with WebRTC sample application. All of the information from the previous run should load.

4. Scroll down and choose **Start Viewer** to connect to the signaling channel as the `viewer` role.

   You should see the media being exchanged between the `master` and `viewer`.

Stream peer-to-peer with WebRTC Ingestion from the sample page to the sample page

1. Follow the section called “Ingest media from a browser” to connect a master participant and make sure it is connected to the storage session.

2. Follow the section called “Add viewers to the ingestion session” to add viewer participants.

   Viewer participants will connect to and receive media from the storage session. They can send optional audio back to the storage session.

   The storage session handles mixing the media received from master and viewer participants and sending it to the appropriate destinations.

3. You can view and consume ingested media through Kinesis Video Streams playback.

Edit the sample application

To edit the SDK and sample application for development purposes, follow the instructions below.

Prerequisite

NodeJS version 16+

> **Note**
>
> We recommend downloading the latest long term support (LTS) version from https://nodejs.org/en/download.

Edit the sample application

1. Download the Kinesis Video Streams with WebRTC SDK in JavaScript.

   Type the following in the terminal:

   ```
   git clone https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-js.git
   ```

2. Navigate to the directory with the package.json file. The file is located in the repository's root directory.

   Type the following in the terminal:

   ```
   cd amazon-kinesis-video-streams-webrtc-sdk-js
   ```

3. Install dependencies.

   Type the following npm CLI command in the terminal:

   ```
   npm install
   ```

4. Start the web server to start serving web pages.

   Type the following npm CLI command in the terminal:

   ```
   npm run develop
   ```

5. In your browser, visit http://localhost:3001/.

   You can make edits to the web page by editing the files in the `examples` directory.

Amazon Kinesis Video Streams WebRTC SDK for Android

The following step-by-step instructions describe how to download, build, and run the Kinesis Video Streams with WebRTC SDK for Android and its corresponding samples.

Note

Amazon Kinesis Video Streams doesn't support IPv6 addresses on Android. See more information about disabling IPv6 on your Android device.

Download the SDK

To download the WebRTC SDK in Android, run the following command:

```
$ git clone https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-
android.git
```

Build the SDK

To build the WebRTC SDK in Android, complete the following steps:

1. Import the Android WebRTC SDK into the Android Studio integrated development environment (IDE) by opening `amazon-kinesis-video-streams-webrtc-sdk-android/build.gradle` with **Open as Project**.
2. If you open the project for the first time, it automatically syncs. If not - initiate a sync. When you see a build error, choose to install any required SDKs by choosing **Install missing SDK package(s)**, then choose **Accept** and complete the install.
3. Configure Amazon Cognito (user pool and identity pool) settings. For details steps, see Configure Amazon Cognito for the SDK. This generates authentication and authorization settings required to build the Android WebRTC SDK.
4. In your Android IDE, open `awsconfiguration.json` (from `src/main/res/raw/`). The file looks like the following:

```
{
  "Version": "1.0",
  "CredentialsProvider": {
    "CognitoIdentity": {
      "Default": {
        "PoolId": "REPLACE_ME",
        "Region": "REPLACE_ME"
      }
    }
  },
  "IdentityManager": {
    "Default": {}
  },
  "CognitoUserPool": {
    "Default": {
      "AppClientSecret": "REPLACE_ME",
      "AppClientId": "REPLACE_ME",
      "PoolId": "REPLACE_ME",
      "Region": "REPLACE_ME"
    }
  }
}
```

Update `awsconfiguration.json` with the values generated by running the steps in Configure Amazon Cognito for the SDK.

5. Make sure your Android device is connected to the computer where you're running the Android IDE. In the Android IDE, select the connected device and then build and run the WebRTC Android SDK.

   This step installs an app called `AWSKinesisVideoWebRTCDemoApp` on your Android device. Using this app, you can verify live WebRTC audio/video streaming between mobile, web and IoT device clients.

Run the sample application

Complete the following steps:

1. On your Android device, open **AWSKinesisVideoWebRTCDemoApp** and log in using either a new (by creating it first) or an existing Amazon Cognito account.
2. In **AWSKinesisVideoWebRTCDemoApp**, navigate to the **Channel Configuration** page and either create a new signaling channel or choose an existing one.

> **Note**
>
> Currently, using the sample application in this SDK, you can only run one signalling channel in **AWSKinesisVideoWebRTCDemoApp**.

3. Optional: choose a unique **Client Id** if you want to connect to this channel as a viewer. Client ID is required only if multiple viewers are connected to a channel. This helps channel's master identify respective viewers.
4. Choose the AWS Region and whether you want to send audio or video data, or both.
5. To verify peer-to-peer streaming, do any of the following:

   > **Note**
   >
   > Ensure that you specify the same signaling channel name, AWS region, viewer ID, and the AWS account ID on all clients that you're using in this demo.

   - Peer-to-peer streaming between two Android devices: master and viewer
     - Using procedures above, download, build, and run the Android WebRTC SDK on two Android devices.
     - Open **AWSKinesisVideoWebRTCDemoApp** on one Android device in master mode (choose **START MASTER**) to start a new session (signaling channel).

       > **Note**
       >
       > Currently, there can only be one master for any given signaling channel.

     - Open **AWSKinesisVideoWebRTCDemoApp** on your second Android device in viewer mode to connect to the signaling channel (session) started in the step above (choose **START VIEWER**).

       Verify that the viewer can see master's audio/video data.
   - Peer-to-peer streaming between the embedded SDK master and an Android device viewer
     - Download, build, and run the Amazon Kinesis Video Streams with WebRTC SDK in C for embedded devices in master mode on a camera device.

    - Using procedures above, download, build, and run the Android WebRTC SDK on an Android device. Open **AWSKinesisVideoWebRTCDemoApp** on this Android device in viewer mode and verify that the viewer can see the embedded SDK master's audio/video data.
- Peer-to-peer streaming between Android device as master and web browser as viewer
    - Using procedures above, download, build, and run the Android WebRTC SDK on an Android device. Open **AWSKinesisVideoWebRTCDemoApp** on this Android device in master mode (choose **START MASTER**) to start a new session (signaling channel).
    - Download, build, and run the Amazon Kinesis Video Streams with WebRTC SDK in JavaScript for web applications as viewer and verify that the viewer can see the Android master's audio/video.

Configure Amazon Cognito for the SDK

Prerequisites

- We recommend Android Studio for examining, editing, and running the application code. We recommend using the latest stable version.
- In the sample code, you provide Amazon Cognito credentials.

Follow these procedures to set up an Amazon Cognito user pool and identity pool.

Set up a user pool

To set up a user pool

1. Sign in to the Amazon Cognito console and verify the region is correct.
2. In the navigation on the left choose **User pools**.
3. In the **User pools** section, choose **Create user pool**.
4. Complete the following sections:

    a. **Step 1: Configure sign-in experience** - In the **Cognito user pool sign-in options** section, select the appropriate options.

       Select **Next**.

    b. **Step 2: Configure security requirements** - Select the appropriate options.

Select **Next**.

c. **Step 3: Configure sign-up experience** - Select the appropriate options.

   Select **Next**.

d. **Step 4: Configure message delivery** - Select the appropriate options.

   In the **IAM role selection** field, select an existing role or create a new role.

   Select **Next**.

e. **Step 5: Integrate your app** - Select the appropriate options.

   In the **Initial app client** field, choose **Confidential client**.

   Select **Next**.

f. **Step 6: Review and create** - Review your selections from the previous sections, then choose **Create user pool**.

5. On the **User pools** page, select the pool that you just created.

   Copy the **User pool ID** and make note of this for later. In the `awsconfiguration.json` file, this is `CognitoUserPool.Default.PoolId`.
6. Select the **App integration** tab and go to the bottom of the page.
7. In the **App client list** section, choose the **App client name** you just created.

   Copy the **Client ID** and make note of this for later. In the `awsconfiguration.json` file, this is `CognitoUserPool.Default.AppClientId`.
8. Show the **Client secret** and make note of this for later. In the `awsconfiguration.json` file, this is `CognitoUserPool.Default.AppClientSecret`.

Set up an identity pool

To set up an identity pool

1. Sign in to the Amazon Cognito console and verify the region is correct.
2. In the navigation on the left choose **Identity pools**.
3. Choose **Create identity pool**.
4. Configure the identity pool.

   a. **Step 1: Configure identity pool trust** - Complete the following sections:

      - **User access** - Select **Authenticated access**
      - **Authenticated identity sources** - Select **Amazon Cognito user pool**

      Select **Next**.
   b. **Step 2: Configure permissions** - In the **Authenticated role** section, complete the following fields:

      - **IAM role** - Select **Create a new IAM role**
      - **IAM role name** - Enter a name and make note of it for a later step.

      Select **Next**.
   c. **Step 3: Connect identity providers** - In the **User pool details** section complete the following fields:

      - **User pool ID** - Select the user pool you created earlier.
      - **App client ID** - Select the app client ID you created earlier.

      Select **Next**.
   d. **Step 4: Configure properties** - Type a name in the **Identity pool name** field.

      Select **Next**.
   e. **Step 5: Review and create** - Review your selections in each of the sections, then select **Create identity pool**.

5. On the **Identity pools** page, select your new identity pool.

   Copy the **Identity pool ID** and make note of this for later. In the `awsconfiguration.json` file, this is `CredentialsProvider.CognitoIdentity.Default.PoolId`.
6. Update the permissions for the IAM role.

   a. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
   b. In the navigation on the left, choose **Roles**.
   c. Find and select the role you created above.

> **Note**
>
> Use the search bar, if needed.

d. Select the attached permissions policy.

   Select **Edit**.

e. Select the **JSON** tab and replace the policy with the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cognito-identity:*",
                "kinesisvideo:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

   Select **Next**.

f. Select the box next to **Set this new version as the default** if it isn't already selected.

   Select **Save changes**.

Amazon Kinesis Video Streams WebRTC SDK for iOS

The following step-by-step instructions describe how to download, build, and run the Kinesis Video Streams WebRTC SDK in iOS and its corresponding samples.

Download the SDK

To download the WebRTC SDK in iOS, run the following command:

```
$ git clone https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-ios.git
```

Build the SDK

Complete the following steps:

1. Import the iOS WebRTC SDK into the XCode integrated development environment (IDE) on an iOS computer by opening `KinesisVideoWebRTCDemoApp.xcworkspace` (path: amazon-kinesis-video-streams-webrtc-sdk-ios/Swift/AWSKinesisVideoWebRTCDemoApp.xcworkspace).
2. If you open the project for the first time, it automatically builds. If not, initiate a build.

   You might see the following error:

   ```
   error: The sandbox is not in sync with the Podfile.lock. Run 'pod install' or
    update your CocoaPods installation.
   ```

   If you see this, do the following:

   a. Change your current working directory to `amazon-kinesis-video-streams-webrtc-sdk-ios/Swift` and run the following in the command line:

      ```
      pod cache clean --all
      pod install
      ```

   b. Change your current working directory to `amazon-kinesis-video-streams-webrtc-sdk-ios` and run the following at the command line:

      ```
      $ git checkout Swift/Pods/AWSCore/AWSCore/Service/AWSService.m
      ```

   c. Build again.
3. Configure Amazon Cognito (user pool and identity pool) settings. For details steps, see Configure Amazon Cognito for the SDK. This generates authentication and authorization settings required to build the iOS WebRTC SDK.
4. In your IDE, open the `awsconfiguration.json` file (from `/Swift/KVSiOSApp`). The file looks like the following:

   ```
   {
       "Version": "1.0",
   ```

```
    "CredentialsProvider": {
        "CognitoIdentity": {
            "Default": {
                "PoolId": "REPLACEME",
                "Region": "REPLACEME"
            }
        }
    },
    "IdentityManager": {
        "Default": {}
    },
    "CognitoUserPool": {
        "Default": {
            "AppClientSecret": "REPLACEME",
            "AppClientId": "REPLACEME",
            "PoolId": "REPLACEME",
            "Region": "REPLACEME"
        }
    }
}
```

Update `awsconfiguration.json` with the values generated by running the steps in Configure Amazon Cognito for the SDK.

5. In your IDE, open the `Constants.swift` file (from `/Swift/KVSiOSApp`). The file looks like the following:

```
import Foundation
import AWSCognitoIdentityProvider

let CognitoIdentityUserPoolRegion = AWSRegionType.USWest2
let CognitoIdentityUserPoolId = "REPLACEME"
let CognitoIdentityUserPoolAppClientId = "REPLACEME"
let CognitoIdentityUserPoolAppClientSecret = "REPLACEME"

let AWSCognitoUserPoolsSignInProviderKey = "UserPool"
let CognitoIdentityPoolID = "REPLACEME"

let AWSKinesisVideoEndpoint = "https://kinesisvideo.us-west-2.amazonaws.com"
let AWSKinesisVideoKey = "kinesisvideo"

let VideoProtocols =  ["WSS", "HTTPS"]
let ConnectAsMaster = "connect-as-master"
let ConnectAsViewer = "connect-as-viewer"

let MasterRole = "MASTER"
let ViewerRole = "VIEWER"

let ClientID = "ConsumerViewer"
```

Update `Constants.swift` with the values generated by running the steps in Configure Amazon Cognito for the SDK.

6. Make sure your iOS device is connected to the Mac computer where you're running XCode. In XCode, select the connected device and then build and run the WebRTC iOS SDK.

   This step installs an app called `AWSKinesisVideoWebRTCDemoApp` on your iOS device. Using this app, you can verify live WebRTC audio/video streaming between mobile, web and IoT device clients.

Run the sample application

Complete the following steps:

1. On your iOS device, open **AWSKinesisVideoWebRTCDemoApp** and log in using either a new (by creating it first) or an existing Amazon Cognito account.
2. In **AWSKinesisVideoWebRTCDemoApp**, navigate to the **Channel Configuration** page and either create a new signaling channel or choose an existing one.

   Note

   Currently, using the sample application in this SDK, you can only run one signalling channel in **AWSKinesisVideoWebRTCDemoApp**.

3. (Optional) Choose a unique **Client Id** if you want to connect to this channel as a viewer. Client Id is required only if multiple viewers are connected to a channel. This helps channel's master identify respective viewers.
4. Choose the AWS Region and whether you want to send audio or video data, or both.
5. To verify peer-to-peer streaming, do any of the following:

> **Note**
>
> Ensure that you specify the same signaling channel name, AWS region, viewer ID, and the AWS account ID on all clients that you're using in this demo.

- Peer-to-peer streaming between two iOS devices: master and viewer
  - Using procedures above, download, build, and run the iOS WebRTC SDK on two iOS devices.
  - Open **AWSKinesisVideoWebRTCDemoApp** on one iOS device in master mode (choose **START MASTER**) to start a new session (signaling channel).

    > **Note**
    >
    > Currently, there can only be one master for any given signaling channel.
  - Open **AWSKinesisVideoWebRTCDemoApp** on your second iOS device in viewer mode to connect to the signaling channel (session) started in the step above (choose **START VIEWER**).

    Verify that the viewer can see master's audio/video data.
- Peer-to-peer streaming between the embedded SDK master and an iOS device viewer
  - Download, build, and run the Amazon Kinesis Video Streams with WebRTC SDK in C for embedded devices in master mode on a camera device.
  - Using procedures above, download, build, and run the iOS WebRTC SDK on an iOS device. Open **AWSKinesisVideoWebRTCDemoApp** on this iOS device in viewer mode and verify that the iOS viewer can see the embedded SDK master's audio/video data.
- Peer-to-peer streaming between iOS device as master and web browser as viewer
  - Using procedures above, download, build, and run the iOS WebRTC SDK on an iOS device. Open **AWSKinesisVideoWebRTCDemoApp** on this iOS device in master mode (choose **START MASTER**) to start a new session (signaling channel).
  - Download, build, and run the Amazon Kinesis Video Streams with WebRTC SDK in JavaScript for web applications as viewer and verify that the JavaScript viewer can see the Android master's audio/video.

Configure Amazon Cognito for the SDK

Prerequisites

- We recommend XCode for examining, editing, and running the application code. We recommend the latest version.
- In the sample code, you provide Amazon Cognito credentials.

Follow these procedures to set up an Amazon Cognito user pool and identity pool.

Set up a user pool

To set up a user pool

1. Sign in to the Amazon Cognito console and verify the region is correct.
2. In the navigation on the left choose **User pools**.
3. In the **User pools** section, choose **Create user pool**.
4. Complete the following sections:

    a. **Step 1: Configure sign-in experience** - In the **Cognito user pool sign-in options** section, select the appropriate options.

       Select **Next**.

    b. **Step 2: Configure security requirements** - Select the appropriate options.

       Select **Next**.

    c. **Step 3: Configure sign-up experience** - Select the appropriate options.

       Select **Next**.

    d. **Step 4: Configure message delivery** - Select the appropriate options.

       In the **IAM role selection** field, select an existing role or create a new role.

       Select **Next**.

    e. **Step 5: Integrate your app** - Select the appropriate options.

       In the **Initial app client** field, choose **Confidential client**.

Select **Next**.

f. **Step 6: Review and create** - Review your selections from the previous sections, then choose **Create user pool**.

5. On the **User pools** page, select the pool that you just created.

   Copy the **User pool ID** and make note of this for later. In the `awsconfiguration.json` file, this is `CognitoUserPool.Default.PoolId`.
6. Select the **App integration** tab and go to the bottom of the page.
7. In the **App client list** section, choose the **App client name** you just created.

   Copy the **Client ID** and make note of this for later. In the `awsconfiguration.json` file, this is `CognitoUserPool.Default.AppClientId`.
8. Show the **Client secret** and make note of this for later. In the `awsconfiguration.json` file, this is `CognitoUserPool.Default.AppClientSecret`.

Set up an identity pool

To set up an identity pool

1. Sign in to the Amazon Cognito console and verify the region is correct.
2. In the navigation on the left choose **Identity pools**.
3. Choose **Create identity pool**.
4. Configure the identity pool.

   a. **Step 1: Configure identity pool trust** - Complete the following sections:

      - **User access** - Select **Authenticated access**
      - **Authenticated identity sources** - Select **Amazon Cognito user pool**

      Select **Next**.

   b. **Step 2: Configure permissions** - In the **Authenticated role** section, complete the following fields:

      - **IAM role** - Select **Create a new IAM role**
      - **IAM role name** - Enter a name and make note of it for a later step.

Select **Next**.

c. **Step 3: Connect identity providers** - In the **User pool details** section complete the following fields:

  - **User pool ID** - Select the user pool you created earlier.
  - **App client ID** - Select the app client ID you created earlier.

  Select **Next**.

d. **Step 4: Configure properties** - Type a name in the **Identity pool name** field.

  Select **Next**.

e. **Step 5: Review and create** - Review your selections in each of the sections, then select **Create identity pool**.

5. On the **Identity pools** page, select your new identity pool.

   Copy the **Identity pool ID** and make note of this for later. In the `awsconfiguration.json` file, this is `CredentialsProvider.CognitoIdentity.Default.PoolId`.

6. Update the permissions for the IAM role.

   a. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

   b. In the navigation on the left, choose **Roles**.

   c. Find and select the role you created above.

   Note

   Use the search bar, if needed.

   d. Select the attached permissions policy.

   Select **Edit**.

   e. Select the **JSON** tab and replace the policy with the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cognito-identity:*",
                "kinesisvideo:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Select **Next**.

f. Select the box next to **Set this new version as the default** if it isn't already selected.

   Select **Save changes**.

Client metrics for the C SDK

Applications built with Amazon Kinesis Video Streams with WebRTC are comprised of various moving parts, including networking, signaling, candidates exchange, peer connection, and data exchange. Kinesis Video Streams with WebRTC in C supports various client-side metrics that enable you to monitor and track the performance and usage of these components in your applications. The supported metrics fall into two major categories: custom metrics defined specifically for the Kinesis Video Streams' implementation of signaling and networking, and media and data-related protocol-specific metrics that are derived from the W3C standard. Note that only a subset of the W3C standard metrics is currently supported for Kinesis Video Streams with WebRTC in C.

Topics

- Signaling metrics
- W3C standard metrics supported for C SDK

Signaling metrics

Signaling metrics can be used to understand how the signaling client behaves while your application is running. You can use the `STATUS signalingClientGetMetrics`

(SIGNALING_CLIENT_HANDLE, PSignalingClientMetrics) API to obtain these signaling metrics. Here's an example usage pattern:

```
SIGNALING_CLIENT_HANDLE signalingClientHandle;
SignalingClientMetrics signalingClientMetrics;
STATUS retStatus = signalingClientGetMetrics(signalingClientHandle,
 &signalingClientMetrics);
printf("Signaling client connection duration: %" PRIu64 " ms",
       (signalingClientMetrics.signalingClientStats.connectionDuration /
 HUNDREDS_OF_NANOS_IN_A_MILLISECOND));
```

The Definition of `signalingClientStats` can be found in Stats.h.

The following signaling metrics are currently supported:

Metric	Description	
cpApiCallLatency	Calculate latency for control plane API calls. Calculati on is done using Exponenti al Moving Average (EMA). The associated calls include: describeChannel, createCha nnel, getChannelEndpoint and deleteChannel.	
dpApiCallLatency	Calculate latency for data plane API calls. Calculati on is done using Exponenti al Moving Average (EMA). The associated calls include: getIceConfig.	
signalingClientUptime	This indicates the time for which the client object exists. Every time this metric is invoked, the most recent uptime value is emitted.	

Metric	Description
connectionDuration	If connection is established, this emits the duration for which the connection is alive. Else, a value of 0 is emitted. This is different from signaling client uptime since, connections come and go, but signalingClientUptime is indicative of the client object itself.
numberOfMessagesSent	This value is updated when the peer sends an offer, answer, or an ICE candidate.
numberOfMessagesReceived	Unlike numberOfMessagesSent, this metric is updated for any type of signaling message. The types of signaling messages are available in `SIGNALING_MESSAGE_TYPE` .
iceRefreshCount	This is incremented when getIceConfig is invoked. The rate at which this is invoked is based on the TTL as part of the ICE configuration received. Each time a fresh set of ICE configuration is received, a timer is set to refresh next time, given the validity of the credentials in the configuration minus some grace period.

Metric	Description	
numberOfErrors	The counter is used to track the number of errors generated within the signaling client. Errors generated while getting ICE configuration, getting signaling state, tracking signaling metrics, sending signaling message, and connecting the signaling client to the web socket in order to send/receive messages are tracked.	
numberOfRuntimeErrors	The metric includes errors that are incurred while the core of the signaling client is running. Scenarios like reconnect failures, message receive failures, and ICE configuration refresh errors are tracked here.	
numberOfReconnects	The metric is incremented on every reconnect. This is a useful metric to understand the stability of the network connection in the set up.	

W3C standard metrics supported for C SDK

A subset of the W3C standard metrics is currently supported for the applications built with the WebRTC C SDK. These fall into the following categories:

- Networking:

  - Ice Candidate: these metrics provide information about the selected local and remote candidates for data exchange between the peers. This includes server source of the candidate, IP address, type of candidate selected for the communication, and candidate priority. These metrics are useful as a snapshot report.
  - Ice Server: these metrics are for gathering operational information about the different ICE servers supported. This is useful when trying to understand the server that is primarily being used for communication and connectivity checks. In some instances, it is also useful to examine these metrics if the gathering of candidates fails.
  - Ice Candidate Pair: these metrics are for understanding the number of bytes/packets that are being exchanged between the peers and also time-related measurements.
- Media and data:
  - Remote Inbound RTP: these metrics represent the endpoint perspective of the data stream sent by the sender.
  - Outbound RTP: these metrics provide information about the outgoing RTP stream. They can also be very useful when analyzing choppy streaming or streaming stops.
  - Inbound RTP: these metrics provide information about the incoming media.
  - Data channel metrics: these metrics can help you analyze the number of messages and bytes sent and received over a data channel. The metrics can be pulled by using the channel ID.

You can use the `STATUS rtcPeerConnectionGetMetrics (PRtcPeerConnection, PRtcRtpTransceiver, PRtcStats)` API to gather metrics related to ICE, RTP and the data channel. Here's a usage example:

```
RtcStats rtcStats;
rtcStats.requestedTypeOfStats = RTC_STATS_TYPE_LOCAL_CANDIDATE;
STATUS retStatus = rtcPeerConnectionGetMetrics (pRtcPeerConnection, NULL, &rtcStats);
printf(“Local Candidate address: %s\n”,
 rtcStats.rtcStatsObject.localIceCandidateStats.address);
```

Here's another example that shows usage pattern to get transceiver related stats:

```
RtcStats rtcStats;
PRtcRtpTransceiver pVideoRtcRtpTransceiver;
rtcStats.requestedTypeOfStats = RTC_STATS_TYPE_OUTBOUND_RTP;
STATUS retStatus = rtcPeerConnectionGetMetrics (pRtcPeerConnection,
 pVideoRtcRtpTransceiver, &rtcStats);
```

```
printf(“Number of packets discarded on send: %s\n”,
 rtcStats.rtcStatsObject.outboundRtpStreamStats.packetsDiscardedOnSend);
```

In the above example, if the second argument to rtcPeerConnectionGetMetrics() is NULL, data for the first transceiver in the list is returned.

Definition for rtcStatsObject can be found in Stats.h. and definition for RtcStats can be found in Include.h.

Sample usages of the APIs and the different metrics can be found in the samples directory in the WebRTC C SDK repository and in the Kinesis Video Stream demos repository.

The following W3C standard metrics are currently supported for the applications built with the WebRTC C SDK.

Topics

Networking

ICE Server Metrics:

Metric	Description	
URL	URL of the STUN/TURN server being tracked	
Port	Port number used by the client	
Protocol	Transport protocol extracted from ICE Server URI. If the value is UDP, ICE tries TURN over UDP, else ICE tried TURN over TCP/TLS. If the URI does not contain transport, ICE tries TURN over UDP and	

Metric	Description
	TCP/TLS. In case of STUN server, this field is empty.
Total Requests Sent	The value is updated for every srflx candidate request and while sending binding request from turn candidates.
Total Responses Received	The value is updated every time a STUN binding response is received.
Total Round Trip Time	The value is updated every time an equivalent response is received for a request. The request packet is tracked in a hash map with the checksum as the key.

ICE Candidate Stats: Only the information about the selected candidate (local and remote) is included.

Metric	Description
address	This indicates the IP address of the local and remote candidate.
port	Port number of the candidate
protocol	Protocol used to obtain the candidate. The valid values are UDP/TCP.
candidateType	Type of candidate selected - host, srflx or relay.

Metric	Description	
priority	Priority of the selected local and remote candidate.	
url	Source of the selected local candidate. This gives an indication of if the candidate selected is received from a STUN server or TURN server.	
relayProtocol	If TURN server is used to obtain the selected local candidate, this field indicates what protocol was used to obtain it. Valid values are TCP/UDP.	

ICE Candidate Pair Stats: Only the information about the selected candidate pairs is included.

Metric	Description	
localCandidateId	The ID of the selected local candidate in the pair.	
remoteCandidateId	The ID of the selected remote candidate in the pair.	
state	State of the candidate pair being inspected.	
nominated	Set to TRUE since the stats are being pulled for selected candidate pair.	
packetsSent	Number of packets sent. This is calculated in the `.` call in the `writeFrame` call.	

Metric	Description	
	This information can also be extracted from outgoing RTP Stats, but since Ice candidate pair includes a lastPacke tSent timestamp, it might be useful to calculate number of packets sent between two points in time.	
packetsReceived	This is updated every time the incomingDataHandler is called.	
bytesSent	This is calculated in the `iceAgentSendPacket()` in the `writeFrame()` call. This is useful when calculati ng a bit rate. Currently, this also includes the header and padding since the ICE layer is oblivious to the RTP packet format.	
bytesReceived	This is updated every time the incomingDataHandle r is called. Currently, this also includes the header and padding since the ICE layer is oblivious to the RTP packet format.	

Metric	Description	
lastPacketSentTimestamp	This is updated every time a packet is sent. This can be used in conjunction with the packetsSent and a recorded start time in application to current packet transfer rate.	
lastPacketReceived Timestamp	This is updated on receiving data in `incomingD ataHandler()` . This can be used in conjunction with packetsReceived to deduce the current packet receive rate. The start time has to be recorded at the applicati on layer in the `transceiv erOnFrame()` callback.	
firstRequestTimestamp	Recorded when the very first STUN binding request is sent out successfully in `iceAgentSendStunPa cket()` . This can be used along with lastReque stTimestamp and requestsS ent to find average time between STUN binding requests.	
lastRequestTimestamp	Recorded every time a STUN binding request is sent out successfully in `iceAgentS endStunPacket()` .	

Metric	Description	
lastResponseTimestamp	Recorded every time a STUN binding response is received.	
totalRoundTripTime	Updated when a binding response is received for a request. The request and response are mapped in a hash table based on checksum.	
currentRoundTripTime	Most recent round trip time updated when a binding response is received for a request on the candidate pair.	
requestsReceived	A counter that is updated on every STUN binding request received.	
requestsSent	A counter that is updated on every STUN binding request sent out in `iceAgentSendStunPacket()` .	
responsesSent	A counter that is updated on every STUN binding response sent out in response to a binding request in `handleStunPacket()` .	
responsesReceived	A counter that is updated on every STUN binding response received in `handleStunPacket()` .	

Metric	Description
packetsDiscardedOnSend	Updated when packet sending fails. In other words, this is updated when `iceUtilsSendData()` fails. This isuseful to determine percentage of packets dropped in a specific duration.
bytesDiscardedOnSend	Updated when packet sending fails. In other words, this is updated when `iceUtilsSendData()` fails. This is useful when determining percentage of packets dropped in a specific duration. Note that the counter also includes the header of the packets.

Media

Outbound RTP Stats

Metric	Description
voiceActivityFlag	This is currently part of `RtcEncoderStats` defined in Include.h. The flag is set to TRUE if the last audio packet contained voice. The flag is currently not set in the samples.

Metric	Description
packetsSent	This indicates the total number of RTP packets sent out for the selected SSRC. This is a part of https://www.w3.org/TR/webrtc-stats/#sentrtpstats-dict* and is included as part of outbound stats. This is incremented every time writeFrame() is called.
bytesSent	Total number of bytes excluding RTP header and padding that is sent. This is updated on every writeFrame call.
encoderImplementation	This is updated by the application layer as part of RtcEncoderStats object.
packetsDiscardedOnSend	This field is updated if the ICE agent fails to send the encrypted RTP packet for any reason in the `iceAgentSendPacket` call.
bytesDiscardedOnSend	This field is also updated if the ICE agent fails to send the encrypted RTP packet for any reason in the `iceAgentSendPacket` call.

Metric	Description	
framesSent	This is incremented only if media stream tack type is MEDIA_STREAM_TRACK _KIND_VIDEO.	
hugeFramesSent	This counter is updated for frames that are 2.5 times the average size of frames. The size of the frame is obtained by calculating the fps (based on the last known frame count time and number of frames encoded in a time interval) and using the targetBitrate in RtcEncode rStats set by the application.	
framesEncoded	This counter is updated only for video track after successfu l encoding of the frame. It is updated on every writeFrame call.	
keyFramesEncoded	This counter is updated only for video track after successful encoding of the key frame. It is updated on every writeFrame call.	

Metric	Description	
framesDiscardedOnSend	This is updated when frame sending fails due to `iceAgentSendPacket` call failure. A frame comprises of a group of packets and currently, framesDis cardedOnSend fails if any packet gets discarded on while sending because of an error.	
frameWidth	This ideally represents the frame width of the last encoded frame. Currently , this is set to a value by the application as part of RtcEncoderStats* *and is of not much significance.	
frameHeight	This ideally represents the frame height of the last encoded frame. Currently , this is set to a value by the application as part of RtcEncoderStats and is of not much significance.	
frameBitDepth	This represents the bit depth per pixel width of the last encoded frame. Currently, this is set by the application as part of RtcEncoderStats and translated into outbound stats.	

Metric	Description	
nackCount	This value is updated every time a NACK is received on an RTP packet and a re-attempt to send the packet is made. The stack supports re-transmission of packets on receiving a NACK.	
firCount	The value is updated on receiving a FIR packet (onRtcpPacket->onRtcpFIRPacket). It indicates how often the stream falls behind and has to skip frames in order to catch up. FIR packet is currently not decoded to extract the fields, so, even though the count is set, no action is taken.	
pliCount	The value is updated on receiving a PLI packet (onRtcpPacket->onRtcpPLIPacket). It indicates that some amount of encoded video data has been lost for one or more frames.	
sliCount	The value is updated on receiving a SLI packet (onRtcpPacket->onRtcpSLIPacket). It indicates how often packet loss affects a single frame.	

Metric	Description	
qualityLimitationResolutionChanges	Currently, the stack supports this metric, however, the frame width and height are not monitored for every encoded frame.	
lastPacketSentTimestamp	The timestamp at which the last packet was sent. It is updated on every writeFrame call.	
headerBytesSent	Total number of RTP header and padding bytes sent for this SSRC excluding the actual RTP payload.	
bytesDiscardedOnSend	This is updated when frame sending fails due to iceAgentSendPacket call failure. A frame comprises of a group of packets, which in turn comprises of bytes and currently, bytesDiscardedOnSend fails if any packet gets discarded on while sending because of an error.	

Metric	Description	
retransmittedPacketsSent	The number of packets that are retransmitted on reception of PLI/SLI/NACK. Currently, the stack only counts the packet resent of NACK since PLI and SLI based retransmissions are not supported.	
retransmittedBytesSent	The number of bytes that are retransmitted on reception of PLI/SLI/NACK. Currently, the stack only counts the bytes resent of NACK since PLI and SLI based retransmissions are not supported.	
targetBitrate	This is set in the application level.	
totalEncodedBytesTarget	This is increased by the target frame size in bytes every time a frame is encoded. This is updated using size parameter in Frame structure.	
framesPerSecond	This is calculated based on the time recorded for the last known encoded frame and the number of frames sent within a second.	
totalEncodeTime	This is set to an arbitrary value in the application and is translated to outbound stats internally.	

Metric	Description	
totalPacketSendDelay	This is currently set to 0 since iceAgentSendPacket sends packet immediately.	

Remote inbound RTP Stats:

Metric	Description	
roundTripTime	The value is extracted from the RTCP receiver report on receiving an RTCP packet type 201 (receiver report). The report comprises of details such as last sender report and delay since last sender report to calculate round trip time. Sender reports are generated roughly every 200 milliseco nds comprising of informati on such as number of packets sent and bytes sent that are extracted from outbound stats.	
totalRoundTripTime	Sum of round trip times calculated	
fractionLost	Represents the fraction of RTP packets lost for the SSRC since the previous sender/re ceiver reporfractionLost was sent.	

Metric	Description	
reportsReceived	Updated every time a receiver report type packet is received.	
roundTripTimeMeasurements	Indicates the total number of reports received for the SSRC that contains valid round trip time. However, currently this value is incremented regardless so its meaning is the same as reportsReceived.	

Inbound RTP Stats:

Metric	Description	
packetsReceived	The counter is updated when a packet is received for a specific SSRC.	
jitter	This metric indicates the packet Jitter measured in seconds for the specific SSRC.	
jitterBufferDelay	This metric denotes the sum of time spent by each packet in the jitter buffer.	
jitterBufferEmittedCount	The total number of audio samples or video frames that have come out of the jitter buffer.	
packetsDiscarded	The counter is updated when the Jitter buffer is full and the packet cannot be pushed	

Metric	Description	
	into it. This can be used to calculate percentage of packets discarded in a fixed duration.	
framesDropped	This value is updated when the `onFrameDroppedFunc()` is invoked.	
lastPacketReceivedTimestamp	Represents the timestamp at which the last packet was received for this SSRC.	
headerBytesReceived	The counter is updated on receiving an RTP packet.	
bytesReceived	Number of bytes received. This does not include the header bytes. This metric can be used to calculate the incoming bit rate.	
packetsFailedDecryption	This is incremented when the decryption of the SRTP packet fails.	

Data channel

Data channel metrics:

Metric	Description	
label	Label is the name of the data channel being inspected.	

Metric	Description	
protocol	Since our stack uses SCTP, the protocol is set to a constant SCTP.	
dataChannelIdentifier	The even or odd identifier used to uniquely identify a data channel. This is updated to an odd value if the SDK is the offerer and even value if SDK is the answerer.	
state	State of the data channel when the stats are queried. Currently, the two states supported are RTC_DATA_ CHANNEL_STATE_CONN ECTING (when the channel is created) and RTC_DATA_ CHANNEL_STATE_OPEN (Set in the onOpen() event).	
messagesSent	The counter is updated when the SDK sends messages over the data channel.	
bytesSent	The counter is updated with the bytes in the message that is sent out. This can be used to understand how many bytes are not sent due to failure, that is, to understand the percentage of bytes that are sent.	
messagesReceived	The metric is incremented in the `onMessage()` callback.	

Metric	Description	
bytesReceived	The metric is generated in the `onMessage()` callback.	

Use Amazon Kinesis Video Streams with WebRTC to ingest and store media

Amazon Kinesis Video Streams offers capabilities to stream video and audio in real-time via WebRTC to the cloud for storage, playback, and analytical processing. Customers can use our enhanced WebRTC SDK and cloud APIs to enable real-time streaming, as well as media ingestion to the cloud.

To get started, you can install Amazon Kinesis Video Streams with WebRTC SDK on any security camera or AWS IoT device with a video sensor and use our APIs to enable media streaming with sub 1-second latency, as well as ingestion and storage in the cloud. Once ingested, you can access your data through our easy-to-use APIs. Amazon Kinesis Video Streams enables you to playback video for live and on-demand viewing, as well as quickly build applications that take advantage of computer vision and video analytics through integration with Amazon Rekognition Video and SageMaker AI.

Topics

API operations

Use the following API operations to configure Amazon Kinesis Video Streams WebRTC ingestion:

- DescribeMappedResourceConfiguration

- DescribeMediaStorageConfiguration
- JoinStorageSession
- JoinStorageSessionAsViewer
- UpdateMediaStorageConfiguration

What is Amazon Kinesis Video Streams with WebRTC ingestion and storage?

Amazon Kinesis Video Streams offers capabilities to stream video and audio in real-time via WebRTC to the cloud for storage, playback, and analytical processing. This topic will provide step-by-step instructions to set up and use our WebRTC SDK and cloud APIs to enable both real-time streaming and media ingestion to the cloud. These instructions include guidance for using the AWS Command Line Interface and the Kinesis Video Streams console.

Before you use Amazon Kinesis Video Streams with WebRTC for the first time, see the section called "Set up an AWS account".

Understanding WebRTC ingestion and storage

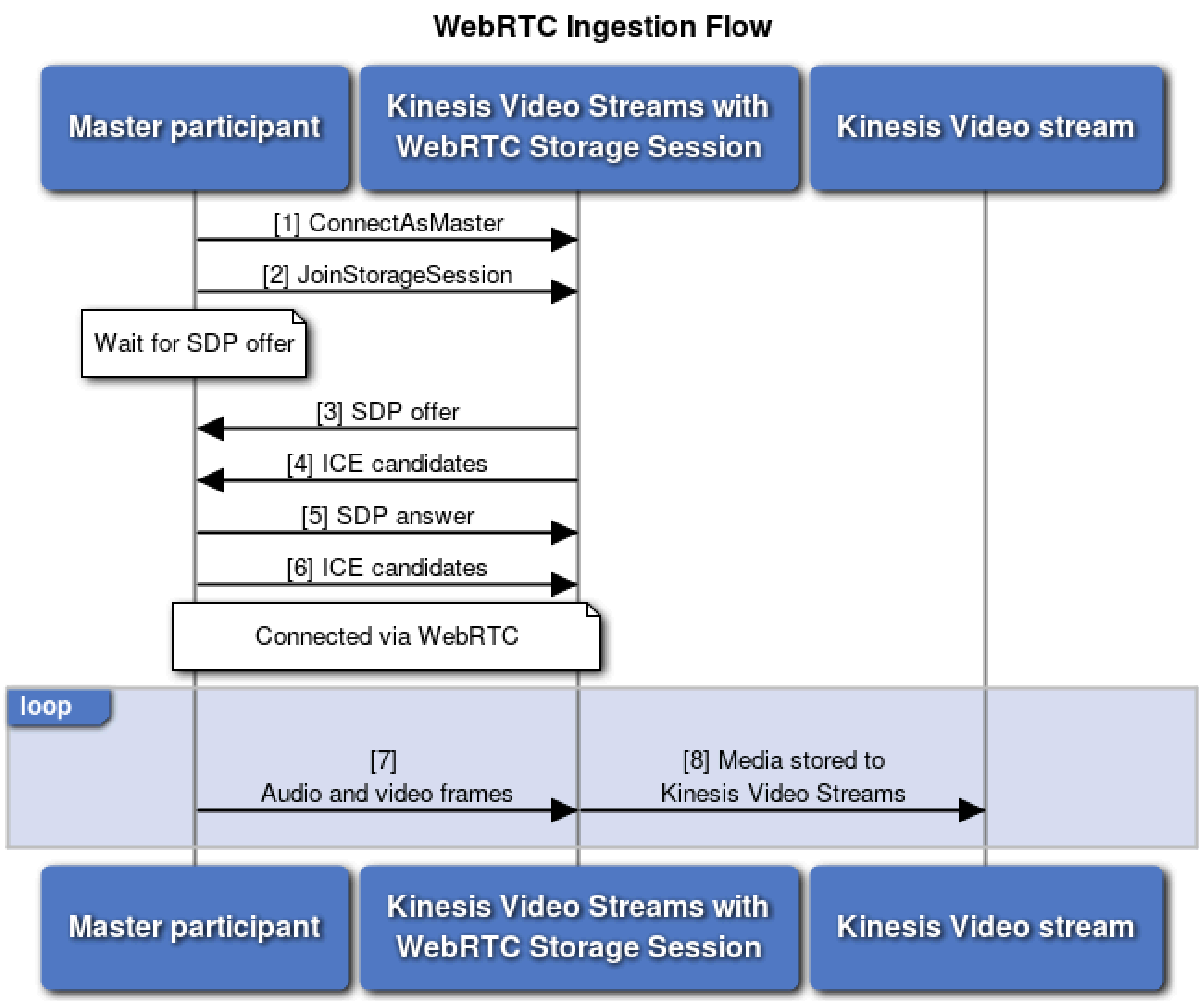
The following sections explain the different ingestion and storage options available in Kinesis Video Streams with WebRTC.

Topics

- Master participant only
- Master and viewer participants together

Master participant only

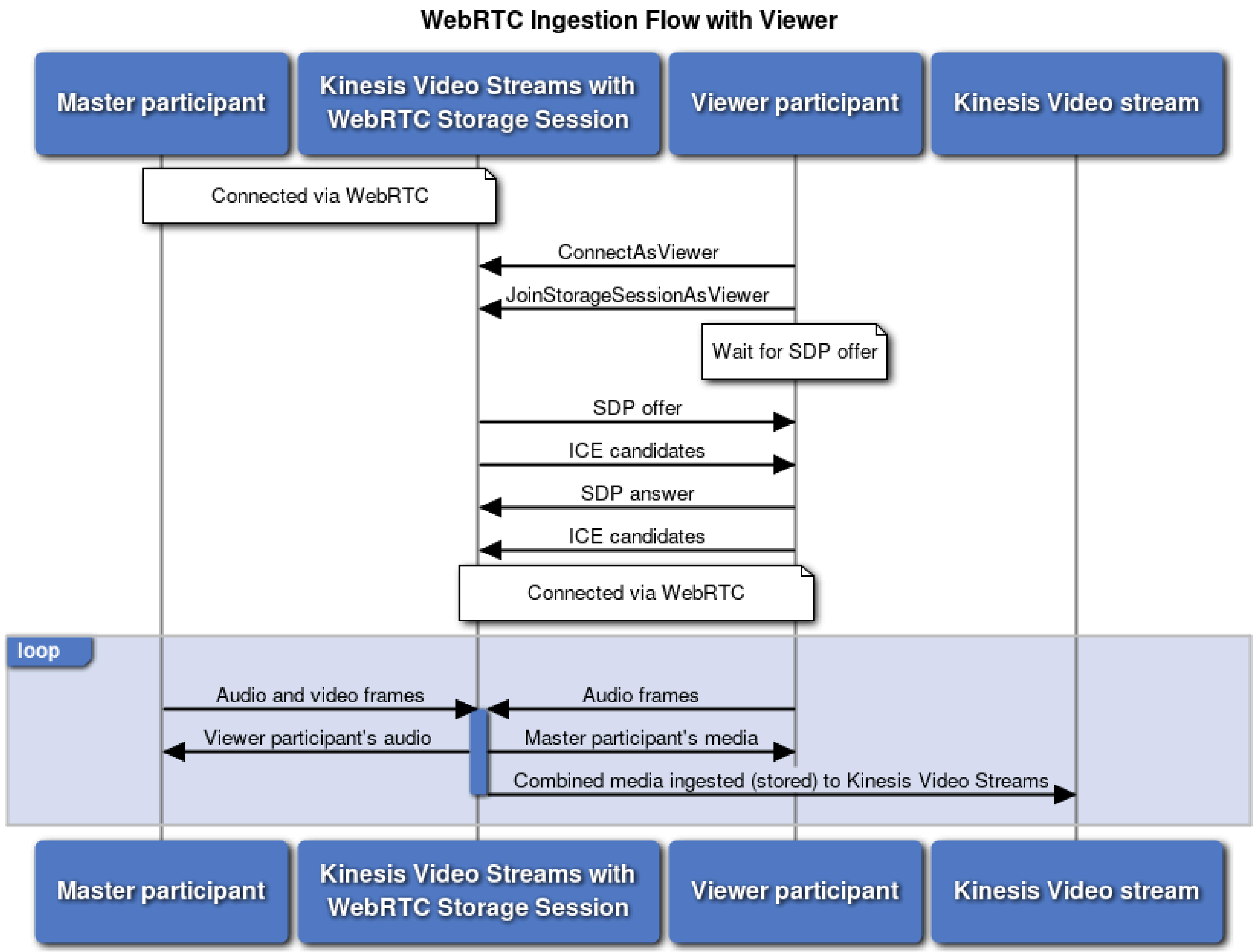
Master participants first connect to Kinesis Video Streams with WebRTC Signaling via the section called "ConnectAsMaster". Next, they call the JoinStorageSession API to have the storage session initiate a WebRTC connection. Once a WebRTC connection is established, media will be ingested to the configured Kinesis video stream.



Master and viewer participants together

Viewer participants first connect to Kinesis Video Streams with WebRTC Signaling via the section called "ConnectAsViewer". Next, they call the JoinStorageSessionAsViewer API to have the storage session initiate a WebRTC connection. Once a WebRTC connection is established, combined media from the master and all viewer participants will be ingested to the configured Kinesis video stream, as long as the master participant is present.

The storage session combines and forwards all viewer participant's audio to the master participant. Viewer participants receive combined media from the master participant and audio from any other viewer participants from the storage session.



Establish a WebRTC connection with the storage session

Since the storage session is within the Amazon network, the storage session will only send `relay` (TURN) candidates to participants. If the participant's network allows, `srflx` (STUN) candidates can be used to connect to the storage session. In other words, from the perspective of the participant, the local nominated ICE candidate can be `srflx` or `relay`, while the remote ICE candidate is always `relay`.

To optimize connection times, don't send `host` candidates to the storage session. The storage session also requires `Trickle ICE` to be used.

See the section called "Troubleshoot storage session connections" to troubleshoot connection issues to the storage session.

Create a signaling channel

A Kinesis Video Streams with WebRTC signaling channel facilitates the exchange of signaling messages required to establish and maintain peer-to-peer connections between WebRTC clients. It handles the negotiation of Session Description Protocol (SDP) offers and answers for session parameters, as well as the exchange of Interactive Connectivity Establishment (ICE) candidates for network information.

To create a signaling channel, call the CreateSignalingChannel API. This page will show you how to invoke that API using the AWS Management Console, AWS CLI, and one of the AWS SDKs.

Important

Make note of the channel ARN, you'll need it later.

AWS Management Console

Do the following:

1. Open the **Kinesis Video Streams Signaling Channels** console at https://console.aws.amazon.com/kinesisvideo/home/#/signalingChannels.
2. Choose **Create signaling channel**.
3. On the **Create a new signaling channel** page, type the name for the signaling channel.

   Leave the default **Time-to-live (Ttl)** value as 60 seconds.

   Choose **Create signaling channel**.
4. Once the signaling channel is created, review the details on the channel's details page.

AWS CLI

Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface User Guide.

For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Run the following Create-Signaling-Channel command using the AWS CLI:

```
aws kinesisvideo create-signaling-channel \
  --channel-name "YourChannelName" \
  --region "us-west-2"
```

The response will look like the following:

```
{
    "ChannelARN": "arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123"
}
```

AWS SDK

This code snippet shows you how to create a Kinesis Video Streams with WebRTC signaling channel using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same. View a complete code example on GitHub.

Create the Kinesis Video Streams client. This is the client used to call the `CreateSignalingChannel` API.

```
const clientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2'
};
const kinesisVideoClient = new AWS.KinesisVideo(clientConfig);
```

Use the client to call the `CreateSignalingChannel` API.

```
const createSignalingChannelResponse = await kinesisVideoClient
    .createSignalingChannel({
        ChannelName: 'YourChannelName',
    })
    .promise();
```

Print the response.

```
console.log(createSignalingChannelResponse.ChannelARN);
```

The live web page with this code sample is available for use on GitHub. Input your region, AWS credentials, and the name of your signaling channel.

Select **Create Channel**.

Create a video stream

Follow these procedures to create a stream that the media will be ingested to. If you have already created the destination stream, skip this step.

Important

WebRTC Ingestion requires a Kinesis video stream with data retention greater than 0. The minimum is 1 hour.

To create a stream, call the CreateStream API using the AWS Management Console, AWS CLI, or one of the AWS SDK's.

Important

Make note of the stream ARN, you'll need it later.

AWS Management Console

Do the following:

1. Open the **Kinesis Video Streams** console at https://console.aws.amazon.com/kinesisvideo/home/.
2. On the **Video streams** page, choose **Create video stream**.
3. On the **Create a new video stream** page, enter *`YourStreamName`* for the stream name. Leave the **Default configuration** button selected.

This will create a stream with a greater than 0 data retention.

Choose **Create video stream**.

4. After Kinesis Video Streams creates the stream, review the details on the *YourStreamName* page.

AWS CLI

Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface User Guide.

For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Run the following `Create-Stream` command using the AWS CLI:

```
aws kinesisvideo create-stream \
  --stream-name "YourStreamName" \
  --data-retention-in-hours 24 \
  --region "us-west-2"
```

The response will look like the following:

```
{
    "StreamARN": "arn:aws:kinesisvideo:us-
west-2:123456789012:stream/YourStreamName/1234567890123"
}
```

AWS SDK

This code snippet shows you how to create a Kinesis video stream using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same. View a complete code example on GitHub.

Create the Kinesis Video Streams client. This is the client used to call the CreateStream API.

```
const clientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2'
};
const kinesisVideoClient = new AWS.KinesisVideo(clientConfig);
```

Use the client to call the `CreateStream` API.

```
const createStreamResponse = await kinesisVideoClient
    .createStream({
        StreamName: 'YourStreamName',
        DataRetentionInHours: 48,
    })
    .promise();
```

Print the response.

```
console.log(createStreamResponse.StreamARN);
```

The live web page with this code sample is available for use on GitHub. Input your region, AWS credentials, and the name of your signaling channel.

Expand the **WebRTC Ingestion and Storage** node, type the name of your stream, then choose **Create Stream**. A pop-up asks for the number of hours you'd like to persist the stream's data. Enter a value greater than 0, then choose **Create Stream**.

Grant permission

You must grant stream permission to your IAM roles in order to ingest streams in Amazon Kinesis Video Streams with WebRTC.

Note

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*.

Both Master and Viewer roles must also have `DescribeStream`, `GetDataEndpoint`, and `PutMedia` permissions to ingest media to Kinesis Video Streams.

Refer to the sample IAM policy below for Master participants:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesisvideo:DescribeSignalingChannel",
        "kinesisvideo:DescribeMediaStorageConfiguration",
        "kinesisvideo:GetSignalingChannelEndpoint",
        "kinesisvideo:GetIceServerConfig",
        "kinesisvideo:ConnectAsMaster",
        "kinesisvideo:JoinStorageSession"
      ],
      "Resource": "arn:aws:kinesisvideo:us-west-2:123456789012:channel/
SignalingChannelName/1234567890123"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kinesisvideo:GetDataEndpoint",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:PutMedia"
      ],
      "Resource": "arn:aws:kinesisvideo:us-west-2:123456789012:stream/
VideoStreamName/1234567890123"
    }
  ]
}
```

Configure destination

Once you've created the Kinesis Video Streams resources, you need to tell the signaling channel which stream to save it to.

If you want to delete a signaling channel or stream, you must unlink them first. See the section called "Unlink signaling channel and stream".

Link signaling channel and stream

Use the UpdateMediaStorageConfiguration API and input the ARNs for the Kinesis Video Streams resources that you want to link.

Important

Once `StorageStatus` is enabled, direct peer-to-peer (master-viewer) connections no longer occur. Peers connect directly to the storage session. You must call the `JoinStorageSession` API to trigger an SDP offer send and establish a connection between a peer and the storage session.

AWS Management Console

Note

This operation isn't currently supported in the Kinesis Video Streams AWS Management Console.

Open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Follow the instructions in the AWS CLI tab.

AWS CLI

Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface documentation.

For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Run the `Update-Media-Storage-Configuration` command in the AWS CLI:

```
aws kinesisvideo update-media-storage-configuration \
  --channel-arn arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123 \
  --media-storage-configuration \
    StreamARN="arn:aws:kinesisvideo:us-
west-2:123456789012:stream/YourStreamName/1234567890123",Status="ENABLED" \
  --region "us-west-2"
```

AWS SDK

This code snippet shows you how to configure the signaling channel to ingest media to the specified Kinesis video stream using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same. View a complete code example on GitHub.

Create the Kinesis Video Streams client. This is the client used to call the UpdateMediaStorageConfiguration API.

```
const clientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2'
};
const kinesisVideoClient = new AWS.KinesisVideo(clientConfig);
```

Use the client to call the `UpdateMediaStorageConfiguration` API.

```
await kinesisVideoClient
    .updateMediaStorageConfiguration({
        ChannelARN: 'YourChannelARN',
        MediaStorageConfiguration: {
            Status: 'ENABLED',
            StreamARN: 'YourStreamARN',
        },
    })
    .promise();
```

The live web page with this code sample is available for use on GitHub. Input your region, AWS credentials, and the name of your signaling channel.

Expand the **WebRTC Ingestion and Storage** node, type the name of your stream, then choose **Update Media Storage Configuration**. The channel will be configured to ingest media to the specified stream.

Unlink signaling channel and stream

Important

You can't delete a signaling channel or stream until they are unlinked from each other.

If you don't want the signaling channel's media ingested to the stream, use the UpdateMediaStorageConfiguration API to unlink the Kinesis Video Streams resources. After the channel is unlinked, direct peer-to-peer connections can resume.

AWS Management Console

Note

This operation isn't currently supported in the Kinesis Video Streams AWS Management Console.

Open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Follow the instructions in the AWS CLI tab.

AWS CLI

Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface documentation.

For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Run the `Update-Media-Storage-Configuration` command in the AWS CLI:

```
aws kinesisvideo update-media-storage-configuration \
  --channel-arn arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123 \
  --media-storage-configuration \
    StreamARN="null",Status="DISABLED" \
  --region "us-west-2"
```

AWS SDK

This code snippet shows you how to configure the signaling channel to ingest media to the specified Kinesis video stream using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same. View a complete code example on GitHub.

Create the Kinesis Video Streams client. This is the client used to call the UpdateMediaStorageConfiguration API.

```
const clientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2'
};
const kinesisVideoClient = new AWS.KinesisVideo(clientConfig);
```

Use the client to call the `UpdateMediaStorageConfiguration` API.

```
await kinesisVideoClient
    .updateMediaStorageConfiguration({
        ChannelARN: 'YourChannelARN',
        MediaStorageConfiguration: {
            Status: 'DISABLED',
            StreamARN: 'null',
        },
    })
    .promise();
```

The live web page with this code sample is available for use on GitHub. Input your region, AWS credentials, and the name of your signaling channel.

Expand the **WebRTC Ingestion and Storage** node, verify that the **Stream Name** field is empty, then choose **Update Media Storage Configuration**. The channel will no longer be configured to ingest media to the specified stream.

Ingest media

The following limits are in place:

- **Session duration:** one hour, maximum
- **Signaling channels:** maximum of 100 per account with storage configuration enabled

Topics

Ingest media from a browser

Important

Chrome is currently the only supported browser.

1. Open the Amazon Kinesis Video Streams with WebRTC SDK in the JavaScript sample page.
2. Complete the following information:
   - **KVS Endpoint** - In the **Region** field, select your region.

     For example, `us-west-2`.
   - **AWS Credentials**

     Complete the following fields:
     - **Access Key ID**
     - **Secret Access Key**

    - **Session Token** - The sample application supports both temporary and long-term credentials. Leave this field blank if you're using long-term IAM credentials. See Temporary security credentials in IAM for more information.
  - **Signaling Channel** - In the **Channel Name** field, type the name of the signaling channel you configured earlier. For more information, see the section called "Configure destination".
  - **Tracks** - Select **Send Video** and **Send Audio**.
  - **WebRTC Ingestion and Storage** - Expand the node and select **Automatically determine ingestion mode**. This option makes the sample application call the DescribeMediaStorageConfiguration API to determine in which mode to run.

3. Select **Start Master**.

   If the signaling channel is configured for ingestion using the DescribeMediaStorageConfiguration API, the sample application will automatically invoke the JoinStorageSession API immediately after connecting to the Signaling channel to start the WebRTC ingestion workflow.

Ingest media from WebRTC C SDK

Follow the the section called "C SDK for embedded devices" procedures to build the sample application.

1. Setup your environment with your AWS account credentials:

   ```
   export AWS_ACCESS_KEY_ID=YourAccessKey
   export AWS_SECRET_ACCESS_KEY=YourSecretKey
   export AWS_DEFAULT_REGION=YourAWSRegion
   ```

   If you're using temporary AWS credentials, also export your session token:

   ```
   export AWS_SESSION_TOKEN=YourSessionToken
   ```

2. Run the samples:

   Master sample

   Navigate to the `build` folder and use "1" as the second argument. Type:

```
./samples/kvsWebrtcClientMaster channel-name 1
```

GStreamer master sample

Navigate to the `build` folder and use "audio-video-storage" as the second argument. Type:

```
./samples/kvsWebrtcClientMasterGstSample channel-name audio-video-storage testsrc
```

This starts WebRTC ingestion.

Note

Your supplied signaling channel must be configured for storage. Use the DescribeMediaStorageConfiguration API to confirm.

Add viewers to the ingestion session

Once the signaling channel is in WebRTC Ingestion mode, viewer participants no longer connect directly to the master participant. Viewer participants connect directly to the storage session. Viewer participants receive the media sent by the master participants, and viewer participants can send back optional audio to the master participant. Any audio sent back by viewers will be sent to all other peers connected to the storage session and ingested to the Kinesis Video Stream, as long as the master participant is connected to the storage session.

The following limits are in place:

- **Maximum number of viewers:** 3
- **Maximum time viewer participants can be connected to the storage session without a master participant present:** 3 minutes

Important

If a viewer disconnects from the storage session (closes the peer connection), their quota (viewer limit) remains consumed for 1 minute. During this 1-minute period, the viewer can invoke this API with the same Client ID to rejoin the session without consuming an additional viewer quota. After 1 minute, the viewer quota is released and available for other viewers to join.

Browser

Important

Chrome is the only supported browser.

1. Open another tab in the Amazon Kinesis Video Streams with WebRTC SDK in the JavaScript sample page. All the information from the previous page will be automatically populated. If not, complete the following information:
   - **KVS Endpoint** - In the **Region** field, select your region.

     For example, `us-west-2`.
   - **AWS Credentials**

     Complete the following fields:
     - **Access Key ID**
     - **Secret Access Key**
     - **Session Token** - The sample application supports both temporary and long-term credentials. Leave this field blank if you're using long-term IAM credentials. See Temporary security credentials in IAM for more information.
   - **Signaling Channel** - In the **Channel Name** field, type the name of the signaling channel you configured earlier. For more information, see the section called "Configure destination".
   - **Tracks** - Select **Send Audio**. Note that if **Send Video** is checked, it will automatically be unchecked when choosing **Start Viewer**.
   - **WebRTC Ingestion and Storage** - Expand the node and select **Automatically determine ingestion mode**. This option makes the sample application call the DescribeMediaStorageConfiguration API to determine in which mode to run.
2. Select **Start Viewer**.

The application automatically calls the JoinStorageSessionAsViewer API immediately after connecting to the signaling channel to trigger an SDP offer send to the viewer from the session.

Note

With peer-to-peer WebRTC, the viewer participant is the controlling peer and the master participant is the controlled peer. In WebRTC ingestion mode, the storage session is now the controlling peer. After connecting to signaling and invoking JoinStorageSessionAsViewer, the viewer will need to respond to the SDP offer and establish a connection to the storage session through WebRTC.

Note

The storage session will only send TURN candidates. When nominating an ICE candidate pair from the perspective of participants, the remote candidate will always be of type `relay`.

Playback ingested media

You can consume media data by either viewing it in the console, or by creating an application that reads media data from a stream using Hypertext Live Streaming (HLS).

View media in the console

In another browser tab, open the AWS Management Console. In the Kinesis Video Streams Dashboard, select Video streams.

Select the name of your stream in the list of streams. Use the search bar, if necessary.

Expand the **Media playback** section. If the video is still uploading, it will be shown. If the upload has finished, select the double-left arrow.

Consume media data using HLS

You can create a client application that consumes data from a Kinesis video stream using HLS. For information about creating an application that consumes media data using HLS, see Kinesis Video Streams playback.

View media using the sample media viewer application

Amazon Kinesis Video Streams created a sample web page implementation of a media viewer capable of playing back Kinesis Video Streams in HLS or DASH. You can view the source code on GitHub and try using the hosted web page.

1. Open the Amazon Kinesis Video Streams Media Viewer.
2. Complete the following fields:
   - **Region** - Select **us-west-2**.
   - **AWS Access Key**
   - **AWS Secret Key**
   - **Stream name**
   - **Playback Mode** - Select **Live**.
3. Select **Start Playback**.

Connect to the storage session

Follow these procedures to create the storage session and start the WebRTC connection process. The master participant should call JoinStorageSession. Viewer participants should call JoinStorageSessionAsViewer.

This will have the storage session send an SDP offer and ICE candidates through signaling to the master participant connected through the section called “ConnectAsMaster”, or the specified viewer participant connected through the section called “ConnectAsViewer”.

1. Obtain the ARN of the signaling channel, as it is a required input for the next step. If you already know the ARN, continue with the next step.

   AWS Management Console

   1. Open the Kinesis Video Streams Signaling Channels console.

2. Choose the name of your signaling channel.
3. Under the **Signaling channel info** tab, locate the ARN for your signaling channel.

AWS CLI

Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface User Guide.

For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Run the following Describe-Signaling-Channel command using the AWS CLI:

```
aws kinesisvideo describe-signaling-channel \
  --channel-name "YourChannelName" \
```

The response will look like the following:

```
{
    "ChannelInfo": {
        "ChannelName": "YourChannelName",
        "ChannelARN": "arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123",
        "ChannelType": "SINGLE_MASTER",
        "ChannelStatus": "ACTIVE",
        "CreationTime": "2024-07-07T23:28:24.941000-07:00",
        "SingleMasterConfiguration": {
        "MessageTtlSeconds": 60
    },
    "Version": "Ws0fZvFGXzEpuZ2CE1s9"
    }
}
```

You'll find the signaling channel ARN in the `ChannelInfo` object.

AWS SDK

This code snippet shows you how to create a Kinesis Video Streams with WebRTC signaling channel using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same.

You can view the complete code example for JoinStorageSession or JoinStorageSessionAsViewer.

Create the Kinesis Video Streams client. This is used to call the DescribeSignalingChannel API.

```
const clientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2'
};
const kinesisVideoClient = new AWS.KinesisVideo(clientConfig);
```

Use the client to call the `DescribeSignalingChannel` API.

```
const describeSignalingChannelResponse = await kinesisVideoClient
    .describeSignalingChannel({
        ChannelName: 'YourChannelName',
    })
    .promise();
```

Save the response.

```
const channelARN =
    describeSignalingChannelResponse.ChannelInfo.ChannelARN;
```

2. Obtain the WEBRTC endpoint. Requests to JoinStorageSession or JoinStorageSessionAsViewer for a particular signaling channel must be made to its designated endpoint.

AWS Management Console

> **Note**
>
> This operation isn't currently supported in the Kinesis Video Streams AWS Management Console.

Open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Follow the instructions in the AWS CLI tab.

AWS CLI

Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface documentation.

For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

Run the `Get-Signaling-Channel-Endpoint` command in the AWS CLI:

```
aws kinesisvideo get-signaling-channel-endpoint \
  --channel-arn "arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123" \
  --single-master-channel-endpoint-configuration
 "Protocols=['WEBRTC'],Role=MASTER" \
  --region "us-west-2"
```

The response looks similar to the following:

```
{
    "ResourceEndpointList": [
        {
            "Protocol": "WEBRTC",
            "ResourceEndpoint": "https://w-abcd1234.kinesisvideo.aws-
region.amazonaws.com"
        }
    ]
}
```

AWS SDK

This code snippet shows you how to call the `GetSignalingChannelEndpoint` API for a Kinesis Video Streams with WebRTC signaling channel using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same. View a complete code example for JoinStorageSession or JoinStorageSessionAsViewer.

Create the Kinesis Video Streams client. This is the client used to call the DescribeSignalingChannel API.

If you created a Kinesis Video Streams client earlier to call `DescribeSignalingChannel`, you can reuse the same client.

```
const clientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2'
};
const kinesisVideoClient = new AWS.KinesisVideo(clientConfig);
```

Use the client to call the `GetSignalingChannelEndpoint` API.

```
const getSignalingChannelEndpointResponse = await kinesisVideoClient
    .getSignalingChannelEndpoint({
        ChannelARN: channelARN,
        SingleMasterChannelEndpointConfiguration: {
            Protocols: ['WEBRTC'],
            Role: 'MASTER',
        },
    })
    .promise();
```

Save the response:

```
const webrtcEndpoint =
 getSignalingChannelEndpointResponse.ResourceEndpointList[0].ResourceEndpoint;
```

3. Use the channel ARN and WEBRTC endpoints to make the API call. If the participants are correctly connected to Kinesis Video Streams with WebRTC Signaling via the `ConnectAsMaster` or `ConnectAsViewer` WebSocket APIs, an SDP offer and a stream of ICE candidate messages will be sent along the WebSocket from the storage session to the participants.

   AWS Management Console

   > **Note**
   >
   > This operation isn't currently supported in the Kinesis Video Streams AWS Management Console.

   Open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

   Follow the instructions in the AWS CLI tab.

   AWS CLI

   Verify that you have the AWS CLI installed and configured. For more information, see the AWS Command Line Interface documentation.

   For installation instructions, see the AWS Command Line Interface User Guide. After installation, configure the AWS CLI with credentials and region.

   Alternatively, open the AWS CloudShell terminal, which has the AWS CLI installed and configured. See the AWS CloudShell User Guide for more information.

   Run the `Join-Storage-Session` command for master participants in the AWS CLI using the channel ARN and WEBRTC endpoint from previous steps:

```
aws kinesis-video-webrtc-storage join-storage-session \
  --endpoint-url https://w-abcd1234.kinesisvideo.us-west-2.amazonaws.com \
  --channel-arn arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123 \
  --region "us-west-2"
```

After a successful run, an empty response is returned and nothing is printed.

In the case of a viewer participant, run the following `Join-Storage-Session-As-Viewer` command in the AWS CLI using the channel ARN and WEBRTC endpoint from before:

```
aws kinesis-video-webrtc-storage join-storage-session-as-viewer \
  --endpoint-url https://w-abcd1234.kinesisvideo.us-west-2.amazonaws.com \
  --channel-arn arn:aws:kinesisvideo:us-
west-2:123456789012:channel/YourChannelName/1234567890123 \
  --client-id "ExampleViewerClientID"
  --region "us-west-2"
```

After a successful run, an empty response is returned and nothing is printed.

AWS SDK

This code snippet shows you how to call the `JoinStorageSession` or `JoinStorageSessionAsViewer` API for a Kinesis Video Streams with WebRTC signaling channel using the AWS SDK for JavaScript v2. The syntax will differ from other AWS SDKs, but the general flow will be the same. View a complete code example for JoinStorageSession or JoinStorageSessionAsViewer.

Create the Kinesis Video Streams WebRTC storage client. This is the client used to call `JoinStorageSession` or `JoinStorageSessionAsViewer` and is different than the Kinesis Video Streams client created in earlier steps.

```
const webrtcStorageClientConfig = {
    accessKeyId: 'YourAccessKey',
    secretAccessKey: 'YourSecretKey',
    region: 'us-west-2',
    endpoint: webrtcEndpoint
};
const kinesisVideoWebRTCStorageClient = new
 AWS.KinesisVideoWebRTCStorage(clientConfig);
```

Use the client to call the `JoinStorageSession` API for master participants.

```
await kinesisVideoWebRTCStorageClient
    .joinStorageSession({
        channelArn: channelARN,
    })
    .promise();
```

In the case of a viewer participant, use the client to call the `JoinStorageSessionAsViewer` API.

```
await kinesisVideoWebRTCStorageClient
    .joinStorageSessionAsViewer({
        channelArn: channelARN,
        clientId: "ExampleViewerClientID",
    })
    .promise();
```

The live web page with this code sample is available on GitHub. Input your region, AWS credentials, and the name of your signaling channel. Expand the **WebRTC Ingestion and Storage** node, and uncheck **Automatically determine ingestion mode**.

Switch the manual override to **ON** and select **Show button to manually call JoinStorageSession API** and/or **Show button to manually call JoinStorageSessionAsViewer API**.

When you select **Start Master** or **Start Viewer**, after the application connects to signaling via `ConnectAsMaster` or `ConnectAsViewer`, a button will appear to have the storage session initiate the WebRTC connection with the peer instead of the application calling the API immediately after connecting to signaling.

Troubleshoot issues connecting with the storage session

This section provides guidance on troubleshooting issues related to setting up and configuring storage for recording video streams.

Topics

- If channel is not mapped to a stream, will also throw 400 InvalidArgumentException

Controlling and controlled peers

In WebRTC, the controlling peer initiates the connection to the controlled peer by sending an SDP offer. For peer-to-peer sessions, the viewer participant initiates the connection by sending an offer to the master participant through Signaling. When connecting to the storage session for WebRTC ingestion, the storage session is the controlling peer. For master participants, they still remain the controlled participant. However, viewer participants switch from controlling to controlled.

Upon calling JoinStorageSession or JoinStorageSessionAsViewer, all participants must respond with an SDP answer and exchange ICE candidates with the storage session.

For a sequence diagram, see Understanding WebRTC ingestion and storage.

Important

Signaling messages received from the storage session do not have a senderClientId field in the JSON, which differs from the peer-to-peer master, which always receives senderClientId field with the SDP offer and ICE candidates.

Review supported codecs

When sending an SDP answer and exchanging ICE candidates with the storage session, we recommend including a `correlationId` in the messages. Including a `correlationId` in the messages allows the storage session to return `statusResponse` messages. These messages will contain the `correlationId` of the input message, allowing you to track which message the `statusResponse` belongs to. This allows you to receive immediate feedback about why your SDP answers was rejected.

For more information about `correlationId` and `statusResponse`, see the section called "Asynchronous message reception".

One common reason the storage session might reject the SDP answer is that the storage session wasn't able to accept codecs specified in the answer. A sample `statusResponse` may look like the following:

```
{
  "correlationId": "1700186220273",
  "errorType": "InvalidArgumentException",
  "statusCode": "400",
  "success": false
}
```

As you review the SDP answer contents, review the lines starting with `a=rtpmap` and verify that the codecs match the storage session's supported codecs. Below is a snippet of a sample SDP answer containing opus audio and VP8 video.

```
...
a=rtpmap:111 opus/48000/2
...
a=rtpmap:120 VP8/90000
...
```

See JoinStorageSession for a list of supported codecs.

If channel is not mapped to a stream, will also throw 400 InvalidArgumentException

Review transceiver directions

In the SDP answer, review the directionality of the video and audio transceivers. The lines in the SDP look like this:

```
a=sendrecv
a=recvonly
```

For master participants, the following requirements are in place:

- H.264 video: sendonly
- Opus audio: sendonly or sendrecv

For viewer participants, the following requirements are in place:

- H.264 video: recvonly
- Opus audio: recvonly or sendrecv

If the service requirements are not met, a statusResponse with 400 IllegalArgumentException will be returned if the correlationId is provided when the answer was sent.

When using the KVS-provided WebRTC ingestion sample applications:

- Master participants: Ensure the "send video" and "send audio" settings are both enabled
- Viewer participants: Ensure "send video" setting is disabled

Review ICE candidate conversion

When receiving ICE candidates from the KVS Signaling SDK, the application may need to convert the string into the ICE candidate object.

ICE candidates received from the storage session do not contain an SDPMID property, and do not come with a senderClientId.

Review your application logic for adding the ICE candidates received from remote into your application's RTCPeerConnection object.

Review ICE candidate queuing logic

All ICE candidates received from the storage session need to be added to the RTCPeerConnection via the addIceCandidate API.

Even though the storage session sends the SDP offer before the ICE candidates, due to the asynchronous nature of the Signaling message delivery, the application may receive the ICE candidates before receiving the offer.

In this case, you will need to implement a temporary buffer to hold the ICE candidates.

The logic implemented in the KVS sample applications is as follows:

1. The samples maintain a map of the senderClientId to its RTCPeerConnection.
2. The samples maintain another map of the senderClientId to a list of pending Ice Candidates.
3. When an ICE candidate is received, check the PeerConnection map. If the PeerConnection map does not have a connection yet, add it to the pending list. Otherwise, add the ICE candidate to the PeerConnection.
4. When the offer is received, it creates the RTCPeerConnection and add it to the PeerConnection map. Then, add all ICE candidates from the pending queue to this PeerConnection.

Use IPv6/Dual-Stack endpoints with Amazon Kinesis Video WebRTC

You can configure Amazon Kinesis Video WebRTC to use IPv6 for both control plane and data plane operations. This enables your applications to communicate with Kinesis Video WebRTC services using IPv6 addresses through dual-stack endpoints.

Note

IPv6 support requires specific SDK versions and configuration settings. Ensure that your Kinesis Video WebRTC SDK and Amazon Web Services SDK versions support IPv6 dual-stack endpoints. Dual-stack endpoints support both IPv4 and IPv6 traffic and are available for some services in some Region.

Amazon Kinesis Video WebRTC supports IPv6 through dual-stack endpoints for both master and viewer applications. You can configure your applications to use IPv6/Dual-Stack endpoints for control plane API calls and data plane operations.

Configure the Amazon Web Services SDK for IPv6-Dual-Stack Endpoints

If you're using the Amazon Web Services SDK to call Kinesis Video WebRTC control plane APIs in your production setup, you can enable IPv6 by configuring dual-stack endpoints. The Amazon Web Services SDK provides several standardized methods to enable dual-stack endpoints.

Important

When dual-stack endpoints are enabled, the SDK attempts to use dual-stack endpoints to make network requests. If a dual-stack endpoint doesn't exist for the service or Region, the request fails.

Use environment variables

Set the following environment variable to enable IPv6 dual-stack endpoints:

```
export AWS_USE_DUALSTACK_ENDPOINT=true
```

Use the Amazon Web Services configuration file

Add the following setting to your Amazon Web Services configuration file (~/.aws/config):

```
[default]
use_dualstack_endpoint = true
```

Use JVM system properties (Java and Kotlin SDKs only)

For Java and Kotlin applications, set the following JVM system property:

```
-Daws.useDualstackEndpoint=true
```

Or programmatically in your Java code:

```
System.setProperty("aws.useDualstackEndpoint", "true");
```

SDK support

The following Amazon Web Services SDKs support dual-stack endpoint configuration:

SDK	Supported	Configuration methods
AWS CLI v2	Yes	Environment variable, configuration file
SDK for C++	Yes	Environment variable, configuration file
SDK for Go V2 (1.x)	Yes	Environment variable, configuration file
SDK for Go 1.x (V1)	Yes	Environment variable, configuration file

SDK	Supported	Configuration methods
SDK for Java 2.x	Yes	Environment variable, configuration file, JVM property
SDK for Java 1.x	No	Not supported
SDK for JavaScript 3.x	Yes	Environment variable, configuration file
SDK for JavaScript 2.x	Yes	Environment variable, configuration file
SDK for Kotlin	Yes	Environment variable, configuration file, JVM property
SDK for .NET 4.x	Yes	Environment variable, configuration file
SDK for .NET 3.x	Yes	Environment variable, configuration file
SDK for PHP 3.x	Yes	Environment variable, configuration file
SDK for Python (Boto3)	Yes	Environment variable, configuration file
SDK for Ruby 3.x	Yes	Environment variable, configuration file
SDK for Rust	Yes	Environment variable, configuration file
SDK for Swift	Yes	Environment variable, configuration file

SDK	Supported	Configuration methods
Tools for PowerShell V5	Yes	Environment variable, configuration file
Tools for PowerShell V4	Yes	Environment variable, configuration file

After you configure dual-stack endpoints, the Amazon Web Services SDK automatically uses IPv6 endpoints when calling Kinesis Video WebRTC control plane APIs.

Configure the Kinesis Video WebRTC SDK for IPv6/Dual-Stack Endpoints

The Kinesis Video WebRTC SDK provides dual-stack configuration options for both control plane and data plane operations. These settings work with the Amazon Web Services SDK dual-stack endpoint configuration.

Configure the WebRTC C SDK

To use dual-stack AWS KVS endpoints and attempt to gather IPv6 ICE candidates, set the following environment variable:

```
export KVS_DUALSTACK_ENDPOINTS=ON
```

In dual-stack mode, ICE gathering will attempt to include IPv6 candidates, but compatibility ultimately depends on the local network configuration and the capabilities of the receiving peers.

To disable dual-stack mode, unset the environment variable:

```
unset KVS_DUALSTACK_ENDPOINTS
```

Data plane endpoint resolution

For data plane operations, the Kinesis Video WebRTC SDK uses the GetSignalingChannelEndpoint API to retrieve the appropriate IPv6/Dual-stack data plane endpoint. The SDK automatically requests IPv6/Dual-stack endpoints when IPv6/Dual-stack is configured.

Important

The GetSignalingChannelEndpoint API has been updated to support IPv6 endpoints. Ensure that you're using a compatible SDK version that supports this functionality.

Configure the AWS CLI for IPv6/Dual-Stack

If you're using the AWS CLI for Kinesis Video WebRTC operations (typically for proof-of-concept work), you can enable IPv6 by configuring dual-stack endpoints.

Use an environment variable

```
export AWS_USE_DUALSTACK_ENDPOINT=true
```

Use the Amazon Web Services configuration file

Add the following to your AWS CLI configuration file (~/.aws/config):

```
[default]
use_dualstack_endpoint = true
```

After you configure dual-stack endpoints, the AWS CLI uses IPv6 dual-stack endpoints for all Amazon Web Services calls, including Kinesis Video WebRTC operations.

Considerations

IoT credentials provider

If you're using IoT credentials for authentication:

- IoT credentials endpoints support IPv6
- Configure dual-stack endpoints using the standard Amazon Web Services SDK configuration methods described previously
- The IoT credentials flow is separate from Kinesis Video WebRTC-specific IPv6 configuration

Network requirements

- Ensure that your network infrastructure supports IPv6 connectivity
- Verify that your security groups and network ACLs allow IPv6 traffic
- Test connectivity to Amazon Web Services IPv6 endpoints from your deployment environment
- Dual-stack endpoints are available for some services in some Regions—verify availability for your target Regions

SDK compatibility

- Ensure that you're using a supported Amazon Web Services SDK version (see the compatibility table)
- The Amazon Web Services SDK for Java 1.x doesn't support dual-stack endpoint configuration
- For the SDK for Go 1.x (V1), you must enable loading from the configuration file to use shared configuration file settings

Testing and validation

Before you deploy IPv6-enabled Kinesis Video WebRTC applications to production:

- Test control plane operations (channel creation, deletion, listing)
- Verify data plane operations (STUN, TURN and WebRTC Signaling)
- Verify successful peer-to-peer streaming session establishment
- Validate performance and connectivity in your network environment
- Run canary tests to ensure consistent IPv6 functionality
- Test failover behavior when dual-stack endpoints aren't available

Customers impacted by the upgrade to include IPv6

When you enable IPv6 for Amazon Kinesis Video WebRTC, there are several areas where you might need to update your existing configurations and policies to ensure continued functionality. This section outlines the key areas that require attention when transitioning to IPv6-enabled endpoints.

IAM policies and IP address filtering

If you use source IP address filtering in your IAM user policies, role policies, or resource-based policies, you need to update these policies to include IPv6 address ranges.

Important

Existing IAM policies that use IPv4 CIDR blocks in IpAddress or NotIpAddress conditions will not automatically work with IPv6 addresses. You must explicitly add IPv6 ranges to maintain access control.

Example IAM policy update for IPv6:

```
{
  "Version": "2012-10-17"       ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "kinesisvideo:*",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "192.0.2.0/24",
            "203.0.113.0/24",
            "2001:db8::/32"
          ]
        }
      }
    }
  ]
}
```

Key considerations for IAM policy updates:

- Add IPv6 CIDR blocks alongside existing IPv4 ranges
- Use the aws:SourceIp condition key for both IPv4 and IPv6 addresses
- Test policies in a non-production environment before deploying

- Consider using aws:RequestedRegion as an additional condition for enhanced security

Network security groups and access control lists

If you're running Kinesis Video WebRTC applications on Amazon EC2 instances or other Amazon Web Services services, you need to update your security groups and network ACLs to allow IPv6 traffic.

- **Security groups** – Add inbound and outbound rules for IPv6 CIDR blocks (::/0 for all IPv6 traffic, or specific IPv6 ranges)
- **Network ACLs** – Update subnet-level network ACLs to allow IPv6 traffic on the required ports
- **Route tables** – Ensure that your VPC route tables include routes for IPv6 traffic to reach internet gateways or NAT gateways

Logging and monitoring

IPv6 addresses have a different format than IPv4 addresses, which can impact your logging, monitoring, and analytics systems.

AWS CloudTrail logs

AWS CloudTrail logs will contain IPv6 addresses in the sourceIPAddress field when requests are made over IPv6. Update your log parsing tools and scripts to handle IPv6 address formats.

Example IPv6 address in AWS CloudTrail logs:

```
{
  "sourceIPAddress": "2001:db8::1",
  "eventName": "CreateSignalingChannel",
  "eventSource": "kinesisvideo.amazonaws.com"
}
```

Application logs

If your applications log client IP addresses or perform IP-based analytics, ensure that your logging infrastructure can handle IPv6 addresses:

- Update log parsing regular expressions to match IPv6 format

- Modify database schemas if you store IP addresses with fixed-length fields
- Update analytics queries and dashboards to work with IPv6 addresses
- Consider using IP address normalization libraries for consistent handling

Monitoring and alerting

Update your monitoring and alerting systems to account for IPv6 traffic:

- Amazon CloudWatch metrics and alarms that filter by IP address
- Custom metrics that track IP-based patterns
- Security monitoring tools that analyze traffic patterns
- Geolocation services that map IP addresses to locations

Third-party integrations

Review and update third-party services and tools that integrate with your Kinesis Video WebRTC applications:

- **Content delivery networks (CDNs) –** Ensure CDN configurations support IPv6 if you're using CDNs for video distribution
- **Load balancers –** Configure Application Load Balancers or Network Load Balancers to handle IPv6 traffic
- **DNS services –** Update DNS records to include AAAA records for IPv6 addresses
- **Firewall and security appliances –** Configure network security appliances to allow IPv6 traffic
- **Monitoring tools –** Verify that third-party monitoring and analytics tools support IPv6 address formats

Application code updates

Review your application code for IPv4-specific assumptions that might need updating:

- **IP address validation –** Update input validation to accept IPv6 address formats
- **Database schemas –** Ensure IP address fields can store IPv6 addresses (typically requiring larger field sizes)
- **Configuration files –** Update any hardcoded IPv4 addresses or CIDR blocks

- **Client libraries –** Verify that HTTP clients and networking libraries support IPv6
- **Error handling –** Update error handling to account for IPv6-specific network errors

Testing and validation

Before enabling IPv6 in production, thoroughly test your applications and infrastructure:

- **Connectivity testing –** Verify that all components can communicate over IPv6
- **Performance testing –** Compare IPv6 and IPv4 performance to identify any issues
- **Security testing –** Validate that security controls work correctly with IPv6 traffic
- **Failover testing –** Test behavior when IPv6 connectivity is unavailable
- **Log analysis –** Verify that logging and monitoring systems correctly handle IPv6 addresses
- **Integration testing –** Test all third-party integrations with IPv6 enabled

Migration strategy

Consider implementing a phased approach to IPv6 adoption:

1. **Assessment phase –** Inventory all systems and identify IPv6 readiness
2. **Preparation phase –** Update policies, security groups, and application code
3. **Testing phase –** Enable IPv6 in development and staging environments
4. **Pilot phase –** Enable IPv6 for a subset of production traffic
5. **Full deployment –** Gradually increase IPv6 traffic until fully deployed
6. **Monitoring phase –** Continuously monitor for issues and optimize performance

Troubleshooting

Common issues

- **Connection failures –** Verify IPv6 network connectivity and DNS resolution
- **SDK errors –** Ensure that you're using compatible SDK versions that support dual-stack endpoints
- **Authentication issues –** Confirm that IAM policies and credentials work with IPv6 endpoints

- **Endpoint not available –** If a dual-stack endpoint doesn't exist for the service or Region, requests fail

Verification steps

- Check that AWS_USE_DUALSTACK_ENDPOINT=true is set or use_dualstack_endpoint = true is in your configuration file
- Verify that Kinesis Video WebRTC SDK IPv6 configuration flags are properly set
- Test network connectivity to Amazon Web Services IPv6 endpoints
- Review application logs for IPv6-specific error messages
- Confirm that your Region supports dual-stack endpoints for Kinesis Video WebRTC

Configuration validation

You can verify your dual-stack endpoint configuration by checking:

- **Environment variables:** echo $AWS_USE_DUALSTACK_ENDPOINT
- **Amazon Web Services configuration file:** cat ~/.aws/config **|** grep use_dualstack_endpoint
- **JVM properties (Java):** Check system properties in your application logs

For additional support and troubleshooting, see the AWS documentation or contact AWS.

Multiviewer

Amazon Kinesis Video Streams Multiviewer is a cloud-based WebRTC solution that enables multiple viewers to concurrently join a real-time video streaming session from a single camera device. This feature addresses the bandwidth and computational constraints of edge devices by processing video streams in the cloud rather than requiring the camera to send separate streams to each viewer.

Topics

Overview

Traditional peer-to-peer WebRTC connections require camera devices to send video separately to each viewer, which can quickly overwhelm devices with limited bandwidth or computational capacity. Multiviewer solves this by using a cloud-based "mixer" that:

- Receives a single video stream from the camera device
- Processes and forwards the stream to multiple viewers
- Handles audio mixing for multi-participant conversations
- Preserves edge device compute and bandwidth capacity

Multiviewer is particularly valuable for use cases including:

- **Smart Home Security:** Multiple household members can view camera feeds simultaneously without degrading performance
- **Enterprise Security:** Security teams can monitor feeds concurrently
- **Automotive Monitoring:** Fleet managers and controllers can view vehicle cameras simultaneously

- **Robotics & Drones:** Multiple operators can monitor autonomous systems
- **Education/Proctoring:** Multiple proctors can observe test sessions
- **Telehealth:** Healthcare teams can participate in remote consultations

Requirements and Resources

To use Multiviewer, you need the following resources:

- **Kinesis Video Streams Stream:** A destination for ingestion and storage of the video and audio content
- **WebRTC Signaling Channel:** Enables connection to devices using the KVS WebRTC SDK
- **Media Storage Configuration:** Links the Stream and Channel using the UpdateMediaStorageConfiguration API

Important

Multiviewer is currently only available when coupled with WebRTC Ingestion. Both master or viewer can initiate a WebRTC ingestion session, and the video and audio tracks are simultaneously stored in a Kinesis Video Streams Stream while being distributed to multiple viewers.

Device Requirements:

- Camera devices must support the KVS WebRTC SDK
- Viewer applications must use the KVS WebRTC SDK
- All participants connect to the same signaling channel

Track Requirements:

- Master participants: Both audio and video tracks are required for WebRTC ingestion
- Viewer participants: Can send an optional audio track or no tracks at all. Viewers cannot send video tracks

Setting Up Multiviewer

Follow these steps to configure Multiviewer for your application:

1. **Create Required Resources**

   Create a Kinesis Video Streams Stream and WebRTC Signaling Channel using the console, CLI, or SDKs. See the section called "Create a video stream" and the section called "Create a signaling channel" for detailed instructions.

2. **Link Resources**

   Use the UpdateMediaStorageConfiguration API to link your Stream and Channel. This configuration enables the Multiviewer functionality. See the section called "Configure destination" for implementation details.

3. **Configure Camera Application**

   Implement the camera application using the KVS WebRTC SDK to call the JoinStorageSession API. This initiates the ingestion session that other viewers can join.

4. **Configure Viewer Applications**

   Implement viewer applications using the KVS WebRTC SDK to call the JoinStorageSessionAsViewer API. Multiple viewers can join the same session simultaneously.

Integration with Ingestion

Multiviewer is built on top of the WebRTC Ingestion capability, which was launched in 2023. This integration provides several benefits:

- **Automatic Recording:** All multiviewer sessions are automatically recorded to Kinesis Video Streams Streams for later playback and analysis
- **Cloud Processing:** Video processing occurs in the cloud, reducing the computational load on edge devices
- **Scalable Architecture:** The cloud-based approach can handle multiple concurrent multiviewer sessions
- **Consistent Experience:** Viewers receive the same high-quality stream regardless of their network conditions

The workflow for a typical multiviewer session with ingestion is:

1. Camera device calls JoinStorageSession to start ingesting video to the cloud
2. Video stream is processed and stored in the configured Kinesis Video Streams Stream
3. Multiple viewer devices call JoinStorageSessionAsViewer to join the session
4. Cloud mixer distributes the video stream to all connected viewers
5. Audio from all participants is mixed and distributed appropriately

API Operations

Multiviewer uses the same API operations as WebRTC Ingestion. The key APIs include:

- UpdateMediaStorageConfiguration - Links a signaling channel to a stream for ingestion
- JoinStorageSession - Initiates an ingestion session from the camera device
- JoinStorageSessionAsViewer - Allows viewers to join an active ingestion session
- DescribeMediaStorageConfiguration - Retrieves the current media storage configuration

For detailed API usage examples, see the section called “Establish a WebRTC connection with the storage session”.

Best Practices

Follow these best practices when implementing Multiviewer:

- **Optimize for Edge Devices:** Use Multiviewer specifically when you need more than 2-3 concurrent viewers, as this is where edge device limitations typically become apparent
- **Monitor Session Health:** Implement monitoring to track session quality and viewer connections
- **Handle Connection Failures:** Implement retry logic for both camera and viewer connections
- **Audio Management:** Consider muting viewers by default to prevent audio feedback in large sessions
- **Resource Cleanup:** Ensure proper cleanup of WebRTC connections when viewers leave sessions

Security

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

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS compliance programs. To learn about the compliance programs that apply to Kinesis Video Streams, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Kinesis Video Streams with WebRTC. The following topics show you how to configure Amazon Kinesis Video Streams with WebRTC to meet your security and compliance objectives. You'll also learn how to use other AWS services that can help you to monitor and secure your Amazon Kinesis Video Streams with WebRTC resources.

Topics

Control access to Amazon Kinesis Video Streams with WebRTC resources with AWS Identity and Access Management

By using AWS Identity and Access Management (IAM) with Amazon Kinesis Video Streams with WebRTC, you can control whether users in your organization can perform a task using specific Kinesis Video Streams with WebRTC API operations and whether they can use specific AWS resources.

For more information about IAM, see the following:

- AWS Identity and Access Management (IAM)
- Getting started with IAM
- IAM User Guide

Contents

Policy syntax

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as follows:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
      }
```

```
        }
    ]
}
```

There are various elements that make up a statement:

- **Effect:** The *effect* can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.
- **Action**: The *action* is the specific API action for which you are granting or denying permission.
- **Resource**: The resource that's affected by the action. To specify a resource in the statement, you need to use its Amazon Resource Name (ARN).
- **Condition**: Conditions are optional. They can be used to control when your policy is in effect.

As you create and manage IAM policies, you might want to use the IAM Policy Generator and the IAM Policy Simulator.

API actions

In an IAM policy statement, you can specify any API action from any service that supports IAM. For Kinesis Video Streams with WebRTC, use the following prefix with the name of the API action: `kinesisvideo:`. For example: `kinesisvideo:CreateSignalingChannel`, `kinesisvideo:ListSignalingChannels`, and `kinesisvideo:DescribeSignalingChannel`.

To specify multiple actions in a single statement, separate them with commas as follows:

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

You can also specify multiple actions using wildcards. For example, you can specify all actions whose name begins with the word "Get" as follows:

```
"Action": "kinesisvideo:Get*"
```

To specify all Kinesis Video Streams operations, use the asterisk (*) wild card as follows:

```
"Action": "kinesisvideo:*"
```

For the complete list of Kinesis Video Streams API actions, see the *Kinesis Video Streams API reference*.

Amazon Resource Names (ARNs)

Each IAM policy statement applies to the resources that you specify using their ARNs.

Use the following ARN resource format for Kinesis Video Streams:

```
arn:aws:kinesisvideo:region:account-id:channel/channel-name/code
```

For example:

```
"Resource": arn:aws:kinesisvideo::*:111122223333:channel/my-channel/0123456789012
```

You can get the ARN of a channel using DescribeSignalingChannel.

Grant other IAM accounts access to a Kinesis video stream

You might need to grant permission to other IAM accounts to perform operations on Kinesis Video Streams with WebRTC signaling channels. 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*.

Example policies

The following example policies demonstrate how you can control user access to your Kinesis Video Streams with WebRTC channels.

Example 1: Allow users to get data from any signaling channel

This policy allows a user or group to perform the `DescribeSignalingChannel`, `GetSignalingChannelEndpoint`, `ListSignalingChannels`, and `ListTagsForResource` operations on any signaling channel.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:Describe*",
                "kinesisvideo:Get*",
                "kinesisvideo:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 2: Allow a user to create a signaling channel

This policy allows a user or group to perform the `CreateSignalingChannel` operation.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:CreateSignalingChannel"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 3: Allow a user full access to all Kinesis Video Streams and Kinesis Video Streams with WebRTC resources

This policy allows a user or group to perform any Kinesis Video Streams operation on any resource. This policy is appropriate for administrators.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesisvideo:*",
            "Resource": "*"
        }
    ]
}
```

Example 4: Allow a user to get data from a specific signaling channel

This policy allows a user or group to get data from a specific signaling channel.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesisvideo:DescribeSignalingChannel",
            "Resource": "arn:aws:kinesisvideo:us-west-2:123456789012:channel/
channel_name/0123456789012"
        }
    ]
}
```

Compliance validation for Amazon Kinesis Video Streams with WebRTC

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 Amazon Kinesis Video Streams with WebRTC

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. You can use Availability Zones to design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in Kinesis Video Streams with WebRTC

As a managed service, Kinesis Video Streams (including its WebRTC capability) is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Kinesis Video Streams through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. We recommend TLS 1.3 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Security best practices for Amazon Kinesis Video Streams with WebRTC

Amazon Kinesis Video Streams (including its WebRTC capability) provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

For security best practices for your remote devices, see Security Best Practices for Device Agents.

Implement least privilege access

When granting permissions, you decide who is getting what permissions to which Kinesis Video Streams resources. You enable specific actions that you want to allow on those resources. Therefore you should grant only the permissions that are required to perform a task. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

For example, a producer that sends data to Kinesis Video Streams requires only `PutMedia`, `GetStreamingEndpoint`, and `DescribeStream`. Do not grant producer applications permissions for all actions (`*`), or for other actions such as `GetMedia`.

For more information, see Apply least-privilege permissions.

Use IAM roles

Producer and client applications must have valid credentials to access Kinesis video streams. You should not store AWS credentials directly in a client application or in an Amazon S3 bucket. These are long-term credentials that are not automatically rotated and could have a significant business impact if they are compromised.

Instead, you should use an IAM role to manage temporary credentials for your producer and client applications to access Kinesis video streams. When you use a role, you don't have to use long-term credentials to access other resources.

For more information, see the following topics in the *IAM user guide*:

- IAM Roles
- Common scenarios for roles: users, applications, and services

Use CloudTrail to monitor API calls

Kinesis Video Streams with WebRTC is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Kinesis Video Streams with WebRTC.

Using the information collected by CloudTrail, you can determine the request that was made to Kinesis Video Streams with WebRTC, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information, see the section called “Log API calls with CloudTrail”.

WebRTC encryption

End to end encryption is a mandatory feature of Amazon Kinesis Video Streams with WebRTC, and Kinesis Video Streams enforces it on all the components, including signaling and media or data streaming. Regardless of whether the communication is peer-to-peer or relayed via Kinesis Video Streams TURN end points, all WebRTC communications are securely encrypted through standardized encryption protocols.

The signaling messages are exchanged using secure Websockets (WSS), data streams are encrypted using Datagram Transport Layer Security (DTLS), and media streams are encrypted using Secure Real-time Transport Protocol (SRTP).

Monitor metrics and API calls

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Kinesis Video Streams with WebRTC and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. Before you start monitoring Kinesis Video Streams with WebRTC, however, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

After you have defined your monitoring goals and have created your monitoring plan, the next step is to establish a baseline for normal Kinesis Video Streams with WebRTC performance in your environment. You should measure Kinesis Video Streams with WebRTC performance at various times and under different load conditions. As you monitor Kinesis Video Streams with WebRTC, you should store a history of monitoring data that you've collected. You can compare current Kinesis Video Streams with WebRTC performance to this historical data to help you to identify normal performance patterns and performance anomalies, and devise methods to address issues that may arise.

Topics

Monitor Kinesis Video Streams with WebRTC

You can monitor an Amazon Kinesis Video Streams with WebRTC using Amazon CloudWatch, which collects and processes raw data from Amazon Kinesis Video Streams with WebRTC into readable, near real-time metrics. These statistics are recorded for a period of 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing.

Amazon Kinesis Video Streams provides the following metrics:

Topics

Signaling metrics

This section provides information on how to monitor and troubleshoot signaling-related issues using CloudWatch Logs.

Metric name	Description	Unit	Dimensions
Failure	'0' is emitted if the Operation mentioned in dimension returns 200 status code response. '1' otherwise.	Count	Operation, SignalingChannelName
Latency	Measures the time it takes for the service to receive a request and return a response.	Milliseconds	Operation, SignalingChannelName
MessagesTransferred.Count	The total number of messages sent and received for a channel.	Count	SignalingChannelName

The `Operation` dimension applies to the following APIs:

- ConnectAsMaster
- ConnectAsViewer

- SendSdpOffer
- SendSdpAnswer
- SendCandidate
- SendAlexaOfferToMaster
- GetIceServerConfig
- Disconnect

TURN metrics

This section provides information on how to monitor and troubleshoot TURN-related issues using CloudWatch Logs.

Metric name	Description	Unit	Dimensions
TURNConnectedMinutes	'1' is emitted for each TURN allocation that is used to stream data through in a minute.	Count	SignalingChannelName

WebRTC ingestion metrics

This section provides information on how to monitor and troubleshoot WebRTC ingestion-related issues using CloudWatch Logs.

Metric name	Description	Unit	Dimensions
Failure	'0' is emitted if the Operation mentioned in dimension returns 200 status code response. '1' otherwise.	Count	Operation, SignalingChannelName
Latency	Measures the time it takes for the service to receive a request and return a response.	Milliseconds	Operation, SignalingChannelName

Metric name	Description	Unit	Dimensions
TotalBitrate	Total bitrate sent. If Role is specified, this represents the total bitrate sent by the MASTER participant or the aggregate total bitrate sent by the VIEWER participant.	bps	Operation , Signaling ChannelNa me, Role
TotalPack etCount	Total packet count sent. If Role is specified, this represents the total bitrate sent by the MASTER participant or the aggregate total bitrate sent by the VIEWER participant.	Count	Operation , Signaling ChannelNa me, Role
WebRTCRec ordingMin utes	The number of WebRTCRecordingMinute occured for a channel.	Count	Operation , Signaling ChannelNa me, Role
WebRTCVie werMinutes	The number of WebRTCViewerMinute occured for a channel.	Count	Operation , Signaling ChannelNa me, Role

The `Operation` dimension applies to the following APIs:

- JoinStorageSession
- JoinStorageSessionAsViewer

`Role` dimension:

- MASTER
- VIEWER

Log API calls with AWS CloudTrail

Amazon Kinesis Video Streams with WebRTC is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Kinesis Video

Streams with WebRTC. CloudTrail captures all API calls for Amazon Kinesis Video Streams with WebRTC as events. The calls captured include calls from the Amazon Kinesis Video Streams console and code calls to the Amazon Kinesis Video Streams with WebRTC API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Kinesis Video Streams with WebRTC. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Kinesis Video Streams with WebRTC, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the *AWS CloudTrail User Guide*.

Amazon Kinesis Video Streams with WebRTC and CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon Kinesis Video Streams with WebRTC, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Kinesis Video Streams with WebRTC, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Region. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Amazon Kinesis Video Streams with WebRTC supports logging the following actions as events in CloudTrail log files:

- CreateSignalingChannel

- DeleteSignalingChannel
- DescribeSignalingChannel
- GetSignalingChannelEndpoint
- ListSignalingChannels
- ListTagsForResource
- TagResource
- UntagResource
- UpdateSignalingChannel

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 or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Example: Log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the CreateSignalingChannel action.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"EX_PRINCIPAL_ID",
      "arn":"arn:aws:iam::123456789012:user/Alice",
      "accountId":"123456789012",
      "accessKeyId":"EXAMPLE_KEY_ID",
      "userName":"Alice"
   },
   "eventTime":"2019-11-19T22:49:04Z",
   "eventSource":"kinesisvideo.amazonaws.com",
   "eventName":"CreateSignalingChannel",
   "awsRegion":"us-west-2",
   "sourceIPAddress":"127.0.0.1",
   "userAgent":"aws-sdk-java/unknown-version Linux/x.xx",
   "requestParameters":{
      "channelName":"YourChannelName"
   },
   "responseElements":{
      "channelARN":"arn:aws:kinesisvideo:us-west-2:123456789012:channel/
YourChannelName/1574203743620"
   },
   "requestID":"df3c99c4-1d97-49da-8569-7de6c92b4856",
   "eventID":"bb74bac2-964c-49b0-903a-3501c6bde632"
}
```

API reference

Topics

WebSocket endpoint APIs

The following are the Amazon Kinesis Video Streams with WebRTC WebSocket endpoint APIs:

Topics

ConnectAsMaster

Connects as a master to the signaling channel specified by the endpoint. Any WebSocket-complaint library can be used to connect to the secure websocket (WSS) endpoint obtained from a `GetSignalingChannelEndpoint` API call. The Amazon Resource Name (ARN) of the signaling channel must be provided as a query string parameter. There are separate endpoints for connecting as a master and as a viewer. If more than one client connects as master to a specific channel, then the most recent request takes precedence. Existing connection metadata is overwritten by the new one.

Request

```
"X-Amz-ChannelARN": "string"
```

- **X-Amz-ChannelARN** - ARN of the signaling channel.
  - Type: string
  - Length constraints: Minimum length of 1. Maximum length of 1024.
  - Pattern: `arn:aws:kinesisvideo:[a-z0-9-]+:[0-9]+:[a-z]+/[a-zA-Z0-9_.-]+/[0-9]+`
  - Required: Yes

Response

200 OK HTTP status code with an empty body.

Errors

- InvalidArgumentException

  A specified parameter exceeds its restrictions, is not supported, or cannot be used. For more information, see the returned message.

  HTTP Status Code: 400
- AccessDeniedException

  The caller is not authorized to access the given channel or the token has expired.

  HTTP Status Code: 403
- ResourceNotFoundException

  The channel doesn't exist.

  HTTP Status Code: 404
- ClientLimitExceededException

  When the API is invoked at a rate that is too high. For more information, see Amazon Kinesis Video Streams with WebRTC service quotas and Error Retries and Exponential Backoff in AWS.

  HTTP Status Code: 400

Limits/Throttling

This API is throttled at an account level if the API is invoked at too high a rate. An error returned when throttled with `ClientLimitExceededException`.

Idempotent

If a connection already exists for the specified clientId and channel, the connection metadata is updated with the new information.

Retry behavior

This is counted as a new API call.

Concurrent calls

Concurrent calls are allowed, the connection metadata is updated for each call.

ConnectAsViewer

Connects as a viewer to the signaling channel specified by the endpoint. Any WebSocket-compliant library can be used to connect to the secure websocket (WSS) endpoint obtained from the `GetSignalingChannelEndpoint` API call. The Amazon Resource Name (ARN) of the signaling channel and the client ID must be provided as query string parameters. There are separate endpoints for connecting as a master and as a viewer. If there is an existing connection with the same `ClientId` as specified in the request, the new connection takes precedence. The connection metadata is overwritten with the new information.

Request

```
"X-Amz-ChannelARN": "string",
"X-Amz-ClientId": "string"
```

- **X-Amz-ChannelARN** - ARN of the signaling channel.
  - Type: string
  - Length constraints: Minimum length of 1. Maximum length of 1024
  - Pattern: `arn:aws:kinesisvideo:[a-z0-9-]+:[0-9]+:[a-z]+/[a-zA-Z0-9_.-]+/[0-9]+`
  - Required: Yes
- **X-Amz-ClientId** - A unique identifier for the client.
  - Type: string
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `^(?!(?i)AWS_.*)[a-zA-Z0-9_.-]`

  > **Note**
  >
  > `X-Amz-ClientId` can't start with `AWS_`.

- Required: Yes

Response

200 OK HTTP status code with an empty body.

Errors

- InvalidArgumentException

  A specified parameter exceeds its restrictions, is not supported, or cannot be used. For more information, see the returned message.

  HTTP Status Code: 400
- AccessDeniedException

  The caller is not authorized to access the given channel or the token has expired.

  HTTP Status Code: 403
- ResourceNotFoundException

  The channel doesn't exist.

  HTTP Status Code: 404
- ClientLimitExceededException

  When the API is invoked at a rate that is too high or when there are more than the supported maximum number of viewers connected to the channel. For more information, see Amazon Kinesis Video Streams with WebRTC service quotas and Error Retries and Exponential Backoff in AWS.

  HTTP Status Code: 400

Limits/Throttling

This API is throttled at an account level if the API is invoked at too high a rate or when there are more than the supported maximum number of viewers connected to the channel. An error returned when throttled with `ClientLimitExceededException`.

Idempotent

If a connection already exists for the specified `ClientId` and channel, the connection metadata is updated with the new information.

Retry behavior

This is counted as a new API call.

Concurrent calls

Concurrent calls are allowed, the connection metadata is updated for each call.

Asynchronous message reception

All response messages are asynchronously delivered to the recipient as events (for example, an SDP offer or SDP answer delivery). The following is the event message structure.

Event

```
{
    "senderClientId": "string",
    "messageType": "string",
    "messagePayload": "string",
    "statusResponse": {
        "correlationId": "string",
        "errorType": "string",
        "statusCode": "string",
        "description": "string"
    }
}
```

- **senderClientId** - A unique identifier for the sender client.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: No
- **messageType** - Type of the event.

  - Type: ENUM
  - Valid Types: `SDP_OFFER`, `SDP_ANSWER`, `ICE_CANDIDATE`, `GO_AWAY`, `RECONNECT_ICE_SERVER`, `STATUS_RESPONSE`
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: Yes
- **messagePayload** - The base64-encoded message content.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 10K.
  - Required: No
- **correlationId** - An unique identifier of the message for which the status is meant. This is the same correlationId provided in the client messages (for example, SDP offer, SDP answer, or ICE candidate).
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: Yes
- **errorType** - A name to uniquely identify the error.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: No
- **statusCode** - HTTP status code corresponding to the nature of the response.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: No
- **description** - A string description explaining the status.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 1K.
  - Required: No

SendSdpOffer

Sends the offer to the target recipient. The prerequisite is that the client must be already connected to the WebSocket endpoint obtained from the `GetSignalingChannelEndpoint` API.

If the sender type is a viewer, then it sends the offer to a master. Also, it is not necessary to specify the `RecipientClientId` and any specified value for `RecipientClientId` is ignored. If the sender type is master, the offer is sent to the target viewer specified by the `RecipientClientId`. `RecipientClientId` is a required input in this case.

A master client app is allowed to send an offer to any viewer, whereas a viewer client app is only allowed to send an offer to a master client app. If a viewer client app attempts to send an offer to another viewer client app, the request will NOT be honored. If there is an outstanding offer for the same client which is not yet delivered, it is overwritten with the new offer.

Request

```
{
    "action": "SDP_OFFER",
    "recipientClientId": "string",
    "messagePayload": "string",
    "correlationId": "string"
}
```

- **action** - Type of the message that is being sent.
  - Type: ENUM
  - Valid values: `SDP_OFFER`, `SDP_ANSWER`, `ICE_CANDIDATE`
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: Yes
- **recipientClientId** - The unique identifier for the recipient.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: Yes
- **messagePayload** - The base-64-encoded message content.

  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 10K.
  - Required: Yes
- **correlationId** - A unique identifier for the message. This is an optional parameter.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: No

Response

If the message is successfully received by the signaling backend, no response is returned. If the service encounters an error and if the `correlationId` is specified in the request, the error details are returned as a `STATUS_RESPONSE` message. For more information, see the section called "Asynchronous message reception".

Errors

- InvalidArgumentException

  A specified parameter exceeds its restrictions, is not supported, or cannot be used. For more information, see the returned message.

  HTTP Status Code: 400
- ClientLimitExceededException

  When the API is invoked at a rate that is too high. For more information, see Amazon Kinesis Video Streams with WebRTC service quotas and Error Retries and Exponential Backoff in AWS.

  HTTP Status Code: 400

Limits/Throttling

This API is throttled at an account level if the API is invoked at too high a rate. An error returned when throttled with `ClientLimitExceededException`.

Idempotent

This API is not idempotent.

Retry behavior

This is counted as a new API call.

Concurrent calls

Concurrent calls are allowed. An offer is sent once per each call.

SendSdpAnswer

Sends the answer to the target recipient. The prerequisite is that the client must be already connected to the WebSocket endpoint obtained from the `GetSignalingChannelEndpoint` API.

If the sender type is a viewer, then it sends the answer to a master. Also, it is not necessary to specify the `RecipientClientId` and any specified value for `RecipientClientId` is ignored. If the sender type is master, the answer is sent to the target viewer specified by the `RecipientClientId`. `RecipientClientId` is a required input in this case.

A master client app is allowed to send an answer to any viewer, whereas a viewer client app is only allowed to send an answer to a master client app. If a viewer client app attempts to send an answer to another viewer client app, the request will NOT be honored. If there is an outstanding answer for the same client which is not yet delivered, it is overwritten with the new answer.

Request

```
{
    "action": "SDP_ANSWER",
    "recipientClientId": "string",
    "messagePayload": "string",
    "correlationId": "string"
}
```

- **action** - Type of the message that is being sent.
  - Type: ENUM
  - Valid values: `SDP_OFFER`, `SDP_ANSWER`, `ICE_CANDIDATE`

  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: Yes
- **recipientClientId** - The unique identifier for the recipient.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: Yes
- **messagePayload** - The base-64-encoded message content.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 10K.
  - Required: Yes
- **correlationId** - A unique identifier for the message.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: No

Response

No response is returned if the message is successfully received by the signaling backend. If the service encounters an error and if the `correlationId` is specified in the request, the error details are returned as a `STATUS_RESPONSE` message. For more information, see Asynchronous message reception.

Errors

- InvalidArgumentException

  A specified parameter exceeds its restrictions, is not supported, or cannot be used. For more information, see the returned message.

  HTTP Status Code: 400
- ClientLimitExceededException

Returned when the API is invoked at a rate that is too high. For more information, see Amazon Kinesis Video Streams with WebRTC service quotas and Error Retries and Exponential Backoff in AWS.

HTTP Status Code: 400

Limits/Throttling

This API is throttled at an account level if the API is invoked at too high a rate. An error is returned when throttled with `ClientLimitExceededException`.

Idempotent

This API is not idempotent.

Retry behavior

This is counted as a new API call.

Concurrent calls

Concurrent calls are allowed. An offer is sent once per each call.

SendIceCandidate

Sends the ICE candidate to the target recipient. The prerequisite is that the client must be already connected to the WebSocket endpoint obtained from the `GetSignalingChannelEndpoint` API.

If the sender type is a viewer, then it sends the ICE candidate to a master. Also, it is not necessary to specify the `RecipientClientId` and any specified value for `RecipientClientId` is ignored. If the sender type is master, the ICE candidate is sent to the target specified by the `RecipientClientId`. `RecipientClientId` is a required input in this case.

A master client app is allowed to send an ICE candidate to any viewer, whereas a viewer client app is only allowed to send an ICE candidate to a master client app. If a viewer client app attempts to send an ICE candidate to another viewer client app, the request will NOT be honored.

Request

```
{
    "action": "ICE_CANDIDATE",
    "recipientClientId": "string",
    "messagePayload": "string",
    "correlationId": "string"
}
```

- **action** - Type of the message that is being sent.
  - Type: ENUM
  - Valid values: `SDP_OFFER`, `SDP_ANSWER`, `ICE_CANDIDATE`
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: Yes
- **recipientClientId** - A unique identifier for the recipient.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: No
- **messagePayload** - The base64-encoded message content.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 10K.
  - Required: Yes
- **correlationId** - A unique identifier for the message.
  - Type: String
  - Length constraints: Minimum length of 1. Maximum length of 256.
  - Pattern: `[a-zA-Z0-9_.-]+`
  - Required: No

Response

No response is returned if the message is successfully received by the signaling backend. If the service encounters an error and if the `correlationId` is specified in the request, the error details are returned as a `STATUS_RESPONSE` message. For more information, see Asynchronous message reception.

Errors

- InvalidArgumentException

  A specified parameter exceeds its restrictions, is not supported, or cannot be used. For more information, see the returned message.

  HTTP Status Code: 400
- ClientLimitExceededException

  When the API is invoked at a rate that is too high. For more information, see Amazon Kinesis Video Streams with WebRTC service quotas and Error Retries and Exponential Backoff in AWS.

  HTTP Status Code: 400

Limits/Throttling

This API is throttled at an account level if the API is invoked at too high a rate. An error returned when throttled with `ClientLimitExceededException`.

Idempotent

This API is not idempotent.

Retry behavior

This is counted as a new API call.

Concurrent calls

Concurrent calls are allowed. An offer is sent once per each call.

Disconnect

A client can close a connection at any time. WebSocket-compliant libraries support close functionality. When the connection is closed, service marks the client as offline for the specific signaling channel and does not try to deliver any messages. The same behavior also applies in the event of idle connection timeout.

The service also sends disconnect indications to the client, for example, during deployments or server maintenance. The following are the defined indication messages:

- **GO_AWAY**: This message is used to initiate the connection shutdown. It enables a client to gracefully process previous messages, disconnect, and reconnect to the signaling channel if needed.
- **RECONNECT_ICE_SERVER**: This message is used to initiate the relay connection shutdown and enables a client to gracefully disconnect, obtain a new ICE server configuration, and reconnect to the relay servers if needed.

Troubleshooting Amazon Kinesis Video Streams with WebRTC

Use the following information to troubleshoot common issues that you might encounter with Amazon Kinesis Video Streams with WebRTC.

Issues establishing a peer-to-peer session

WebRTC can help alleviate issues that occur due to:

- Network address translation (NAT)
- Firewalls
- Proxies between peers

WebRTC provides a framework to help negotiate and maintain connections for as long as the peers are connected. It also provides a mechanism for relaying media through a TURN server in case a peer-to-peer connection can't be negotiated.

Given all of the components necessary to establish the connection, it's worth understanding a few tools that are available to help troubleshoot issues related to establishing a session.

Topics

Session Description Protocol (SDP) offers and answers

Session Description Protocol (SDP) offers and answers initialize the RTC session between peers.

To learn more about the SDP protocol, see the specification.

- **Offers** are generated by "viewers" who want to connect to peers that are connected to the signaling channel as a "master" in Kinesis Video Streams with WebRTC.

- **Answers** are generated by the receiver of the offer.

Both offers and answers are generated on the client side, although they might contain ICE candidates that have been gathered up to that point.

The Kinesis Video Streams WebRTC SDK for C includes a simple environment variable that you can set to log the SDP. This is useful to understand both the offers being received and the answers being generated.

To log SDPs to `stdout` from the SDK, set the following environment variable: `export DEBUG_LOG_SDP=TRUE`. You can also log SDP offers and answers in JavaScript-based clients using the `sdpOffer` event. To see this demonstrated, see GitHub.

For additional information, see the section called "Monitor Kinesis Video Streams with WebRTC".

If the SDP answer is not returned, it is possible that the peer was unable to accept the SDP offer, since the offer does not contain any compatible media codecs. You may see logs similar to the following:

```
I/webrtc_video_engine.cc: (line 808): SetSendParameters: {codecs:
 [VideoCodec[126:H264]], conference_mode: no, extensions: [], extmap-allow-mixed:
 false, max_bandwidth_bps: -1, mid: video1}
E/webrtc_video_engine.cc: (line 745): No video codecs supported.
E/peer_connection.cc: (line 6009): Failed to set remote video description send
 parameters for m-section with mid='video1'. (INVALID_PARAMETER)
E/peer_connection.cc: (line 3097): Failed to set remote offer sdp: Failed to set remote
 video description send parameters for m-section with mid='video1'.
E/KinesisVideoSdpObserver: onSetFailure(): Error=Failed to set remote offer sdp: Failed
 to set remote video description send parameters for m-section with mid='video1'.
D/KVSWebRtcActivity: Received SDP offer for client ID: null. Creating answer
E/peer_connection.cc: (line 2373): CreateAnswer: Session error code: ERROR_CONTENT.
 Session error description: Failed to set remote video description send parameters for
 m-section with mid='video1'..
E/KinesisVideoSdpObserver: onCreateFailure(): Error=Session error code: ERROR_CONTENT.
 Session error description: Failed to set remote video description send parameters for
 m-section with mid='video1'..
```

As you review the SDP offer contents, look for lines starting with `a=rtpmap` to see which media codecs are being requested.

```
...
```

```
a=rtpmap:126 H264/90000
...
a=rtpmap:111 opus/48000/2
...
```

If you're using Safari as a viewer to connect to a master sending H.265 media and you encounter the following:

- `InvalidAccessError: Failed to set remote answer sdp: Called with SDP without DTLS fingerprint.`
- `InvalidAccessError: Failed to set remote answer sdp: rtcp-mux must be enabled when BUNDLE is enabled.`

Confirm the issue is with the SDP offer generated by the browser. In the SDP offer, search for lines starting for `a=rtpmap`, and check if a line for H.265 is present. It should look like this:

```
a=rtpmap:104 H265/90000
```

If it's not present, enable the H.265 codec for WebRTC in the Safari settings.

In the Safari top navigation, do the following:

- Select **Safari** > **Settings...** > **Advanced**. Select the **Show features for web developers** box.
- Select **Feature Flags**. Select the **WebRTC H265 codec** box.

Restart your browser for the changes to take effect.

Evaluate ICE candidate generation

ICE candidates are generated by each client that makes calls to the STUN server. For Kinesis Video Streams with WebRTC, the STUN server is `stun:stun.kinesisvideo.{aws-region}.amazonaws.com:443`.

In addition to calling the STUN server to obtain candidates, clients often also call the TURN servers. They make this call so that the relay server can be used as a fallback in case a direct peer-to-peer connection can't be established.

You can use the following tools to generate ICE candidates:

- A Trickle ICE WebRTC sample, which uses Trickle ICE to gather candidates
- IceTest.Info

With both of these tools. you can enter the STUN and TURN server information to gather candidates.

To obtain the TURN server information and necessary credentials for Kinesis Video Streams with WebRTC, you can call the GetIceServerConfig API operation.

The following AWS CLI calls demonstrate how to obtain this information to use in these two tools.

```
export CHANNEL_ARN="YOUR_CHANNEL_ARN"

aws kinesisvideo get-signaling-channel-endpoint \
    --channel-arn $CHANNEL_ARN \
    --single-master-channel-endpoint-configuration Protocols=WSS,HTTPS,Role=MASTER
```

The output from the `get-signaling-channel-endpoint` command returns a response that looks like this:

```
{
  "ResourceEndpointList": [
    {
      "Protocol": "HTTPS",
      "ResourceEndpoint": "https://your-endpoint.kinesisvideo.us-east-1.amazonaws.com"
    },
    {
      "Protocol": "WSS",
      "ResourceEndpoint": "wss://your-endpoint.kinesisvideo.us-east-1.amazonaws.com"
    }
  ]
}
```

Use the HTTPS `ResourceEndpoint` value to obtain the list of TURN servers as follows:

```
export ENDPOINT_URL="https://your-endpoint.kinesisvideo.us-east-1.amazonaws.com"

aws kinesis-video-signaling get-ice-server-config \
    --channel-arn $CHANNEL_ARN \
    --service TURN \
    --client-id my-amazing-client \
    --endpoint-url $ENDPOINT_URL
```

The response contains TURN server details, including endpoints for TCP and UDP and the credentials required to access them.

> **Note**
>
> The TTL value in the response determines the duration, in seconds, that these credentials are valid for. Use these values in the Trickle ICE WebRTC sample or in IceTest.Info to generate ICE candidates using the Kinesis Video Streams managed service endpoints.

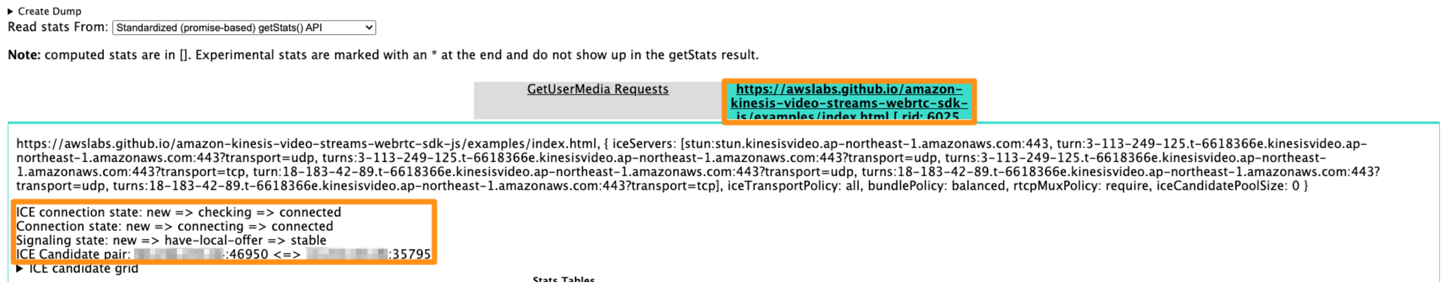
Determine which candidates were used to establish the connection

It can be helpful to understand which candidates were used to successfully establish the session. If you have a browser-based client running an established session, you can determine this information in Google Chrome by using the built-in webrtc-internals utility.

Open a WebRTC session in one browser tab.

In another tab, open `chrome://webrtc-internals/`. You can view all of the information about your ongoing session in this tab.

You will see information about the established connection. For example:

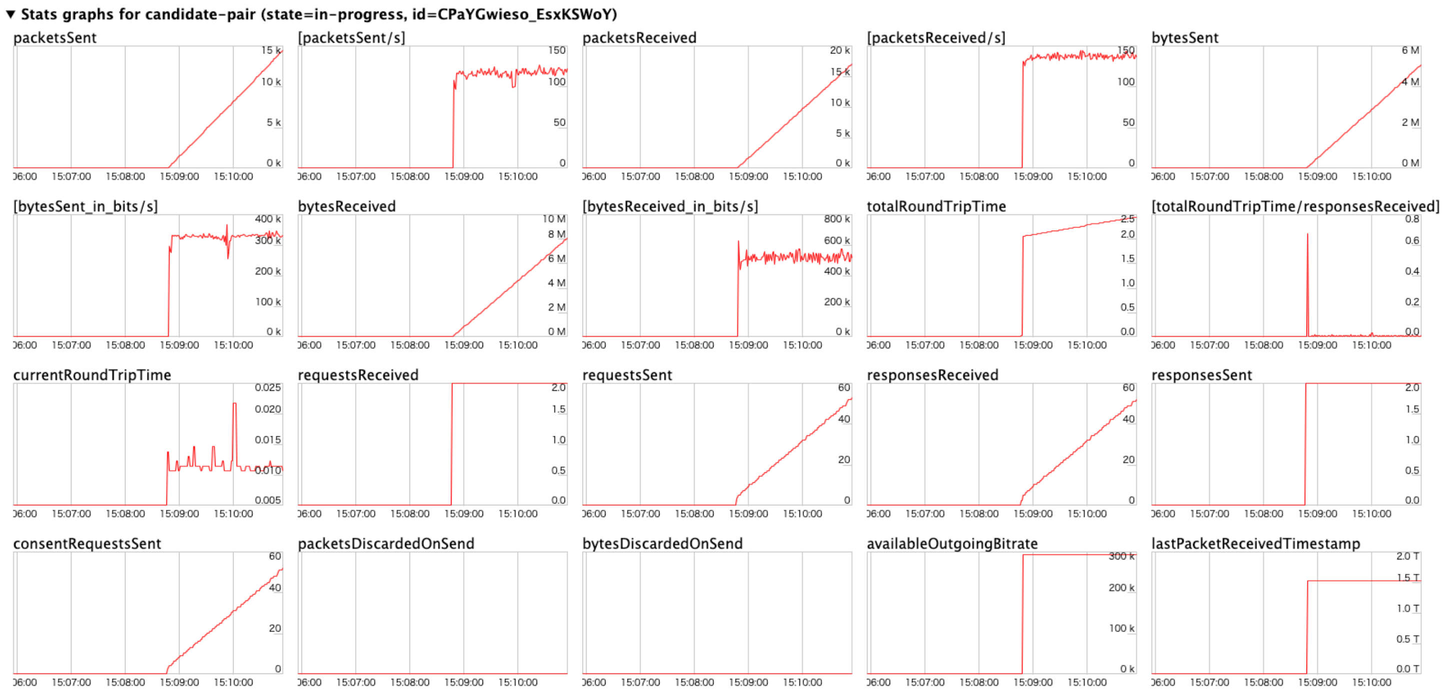

▸ Create Dump
Read stats From: Standardized (promise-based) getStats() API
GetUserMedia Requests
https://awslabs.github.io/amazon-kinesis-video-streams-webrtc-sdk-js/examples/index.html [rid: 6025
▸ ICE candidate grid
Stats Tables

Note: computed stats are in []. Experimental stats are marked with an * at the end and do not show up in the getStats result.

```
https://awslabs.github.io/amazon-kinesis-video-streams-webrtc-sdk-js/examples/index.html, { iceServers: [stun:stun.kinesisvideo.ap-northeast-1.amazonaws.com:443, turn:3-113-249-125.t-6618366e.kinesisvideo.ap-northeast-1.amazonaws.com:443?transport=udp, turns:3-113-249-125.t-6618366e.kinesisvideo.ap-northeast-1.amazonaws.com:443?transport=udp, turns:3-113-249-125.t-6618366e.kinesisvideo.ap-northeast-1.amazonaws.com:443?transport=tcp, turn:18-183-42-89.t-6618366e.kinesisvideo.ap-northeast-1.amazonaws.com:443?transport=udp, turns:18-183-42-89.t-6618366e.kinesisvideo.ap-northeast-1.amazonaws.com:443?transport=udp, turns:18-183-42-89.t-6618366e.kinesisvideo.ap-northeast-1.amazonaws.com:443?transport=tcp], iceTransportPolicy: all, bundlePolicy: balanced, rtcpMuxPolicy: require, iceCandidatePoolSize: 0 }

ICE connection state: new => checking => connected
Connection state: new => connecting => connected
Signaling state: new => have-local-offer => stable
ICE Candidate pair: [redacted]:46950 <=> [redacted]:35795
```

You can also confirm metrics like the following for the established connection.



ICE-related timeouts

Default timeout values are set for ICE in the KvsRtcConfiguration. The defaults should be sufficient for most users, but you may need to adjust them to improve chances of establishing a connection over a poor network. You can configure these defaults in the application.

Review the log for the default settings:

```
2024-01-08 19:43:44.433 INFO    iceAgentValidateKvsRtcConfig():
 iceLocalCandidateGatheringTimeout: 10000 ms
 iceConnectionCheckTimeout: 12000 ms
 iceCandidateNominationTimeout: 12000 ms
 iceConnectionCheckPollingInterval: 50 ms
```

If you have poor network quality and want to improve chances of connection, try adjusting the following values:

- `iceLocalCandidateGatheringTimeout` - Increase this timeout limit to gather additional potential candidates to try for connection. The goal is to try all possible candidate pairs, so if you're on a poor network, increase this limit to give more time to gather.

  For example, if the host candidates didn't work and server reflexive (srflx) or relay candidates need to be tried, you may need to increase this timeout. Due to poor network, the candidates are gathered slowly and the application doesn't want to spend more than 20 seconds on this step. Increasing the timeout provides more time to gather potential candidates to try for connection.

> **Note**
>
> We recommend that this value be less than `iceCandidateNominationTimeout`, since the nomination step needs to have time to work with the new candidates.

- `iceConnectionCheckTimeout` - Increase this timeout in unstable or slow networks, where the packet exchange and binding request/response take time. Increasing this timeout allows at least one candidate pair to be tried for nomination by the other peer.
- `iceCandidateNominationTimeout` - Increase this timeout to ensure candidate pairs with the local relay candidate are tried.

  For example, if it takes about 15 seconds to gather the first local relay candidate, set the timeout to a value more than 15 seconds to ensure candidate pairs with the local relay candidate are tried for success. If the value is set to less than 15 seconds, the SDK would lose out on trying a potential candidate pair, leading to connection establishment failure.

  > **Note**
  >
  > We recommend that this value be greater than `iceLocalCandidateGatheringTimeout`, in order for it to have an effect.

- `iceConnectionCheckPollingInterval` - This value defaults to 50 milliseconds per specification. Changing this value changes the frequency of connectivity checks and, essentially, the ICE state machine transitions.

  In a reliable, high performance network setting with good system resources, you can decrease the value to help in faster connection establishment. Increasing the value could help reduce the network load, but the connection establishment could slow down.

  > **Important**
  >
  > We don't recommend changing this default.

Document history for the Amazon Kinesis Video Streams with WebRTC developer guide

Change	Description	Date
Revised content organization	Revised the layout and organization within the Amazon Kinesis Video Streams with WebRTC Developer Guide.	September 18, 2024
Initial publication	Initial publication of the Amazon Kinesis Video Streams with WebRTC Developer Guide. Learn more	November 4, 2019