Kinesis Video Streams

Amazon Kinesis Video Streams WebRTC Developer Guide

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:

- Kinesis Video Streams with WebRTC: How It Works (p. 3)
- WebRTC SDK in C for Embedded Devices (p. 20)
- WebRTC SDK in JavaScript for Web Applications (p. 22)
- WebRTC SDK for Android (p. 23)
- WebRTC SDK for iOS (p. 26)
- Control plane APIs
- Data plane REST APIs
- Data plane Websocket APIs

Kinesis Video Streams with WebRTC Pricing

For information about Kinesis Video Streams with WebRTC pricing, see Amazon Kinesis Video Streams Pricing.

Accessing 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.

Kinesis Video Streams with WebRTC: How It Works

Topics

Amazon Kinesis Video Streams with WebRTC 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, webcam, 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.

WebRTC 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.

Kinesis Video Streams with WebRTC 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.
  - WebRTC SDK in C for Embedded Devices (p. 20)
  - WebRTC SDK in JavaScript for Web Applications (p. 22)
  - WebRTC SDK for Android (p. 23)
  - WebRTC SDK for iOS (p. 26)

WebRTC Websocket APIs

Topics

ConnectAsViewer

Connects as a viewer to the signaling channel specified by the endpoint. Any WebSocket-compliant library can be used to connect to the endpoint obtained from the `GetSignalingEndpoint` 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 and 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` cannot start with an `AWS_ prefix`.
  - 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 (p. 16) 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.

ConnectAsMaster

Connects as a master to the signaling channel specified by the endpoint. Any WebSocket-complaint library can be used to connect to the 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 (p. 16) 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.

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 Asynchronous Message Reception (p. 14).

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 (p. 16) 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 (p. 14).

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 (p. 16) 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 (p. 14).

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 (p. 16) 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.

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

Amazon Kinesis Video Streams with WebRTC Service Quotas

Kinesis Video Streams with WebRTC has the following service quotas:

The service quotas are either soft [s], which can be upgraded by submitting a support ticket, or hard [h], which can't be increased.

Topics

Control Plane API Service Quotas

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

Control Plane API Service Quotas

API	Account service quota: Request	Account service quota: Channels	Channel-level service quota	Relevant Exceptions and Notes
CreateSignalingChannel	50 TPS [s]	1000 signaling channels per account [s] per region, in all other supported regions.		
DescribeSignalingChannel	50 TPS [h]	N/A	5 TPS [h]	
UpdateSignalingChannel	50 TPS [h]	N/A	5 TPS [h]	
ListSignalingChannels	50 TPS [h]	N/A		
DeleteSignalingChannel	50 TPS [h]	N/A	5 TPS [h]	
GetSignalingChannelEndpoint	500 TPS [h]	N/A		
TagResource	50 TPS [h]	N/A	5 TPS [h]	
UntagResource	50 TPS [h]	N/A	5 TPS [h]	
ListTagsForResource	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 Kinesis Video Streams with WebRTC: How It Works (p. 3).

- ConnectAsMaster
  - API - 3 TPS per channel (hard)
  - Maximum number of master connections per signaling channel - 1 (hard)
  - Connection duration limit - 1 hour (hard)
  - Idle connection timeout - 10 minutes (hard)
  - When a client receives the `GO_AWAY` message from the server, connection is terminated after a grace period of 1 minute (hard)
- ConnectAsViewer
  - API - 3 TPS per channel (hard)
  - Maximum number of viewer connections per channel - 10 (soft)
  - Connection duration limit - 1 hour (hard)
  - Idle connection timeout - 10 minutes (hard)
  - Once a client receives the `GO_AWAY` message from the server, connection is terminated after a grace period of 1 minute (hard)
- SendSDPOffer
  - API: 5 TPS per WebSocket connection (hard)
  - Message payload size limit - 10k (hard)
- SendSDPAnswer
  - API: 5 TPS per WebSocket connection (hard)
  - Message payload size limit - 10k (hard)
- SendICECandidate
  - API: 20 TPS per WebSocket connection (hard)
  - Message payload size limit - 10k (hard)
- SendAlexaOffertoMaster
  - API: 5 TPS per signaling channel (hard)
- GetIceServerConfig
  - API: 5 TPS per signaling channel (hard)
- Disconnect
  - N/A

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 Kinesis Video Streams with WebRTC: How It Works (p. 3).

- Bit Rate - 5Mbps (hard)
- Credential Lifecycle - 5 minutes (hard)
- Number of allocations - 50 per signaling channel (hard)

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.
- Use the Kinesis Video Streams with WebRTC SDKs to configure master and viewer to perform peer-to-peer video and audio streaming over a signaling channel.

If you are new to Kinesis Video Streams with WebRTC, we recommend that you read Kinesis Video Streams with WebRTC: How It Works (p. 3) first.

Topics

Set Up an AWS Account and Create an Administrator

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

1. Sign Up for AWS (p. 18) (unless you already have an account)
2. Create an Administrator IAM User (p. 18)
3. Create an AWS Account Key (p. 19)

Sign Up for AWS

If you already have an AWS account, you can skip this step.

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including Kinesis Video Streams with WebRTC.

To create an AWS account

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

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

Write down your AWS account ID because you need it for the next task.

Create an Administrator IAM User

When you sign up for AWS, you provide an email address and password that is associated with your AWS account. This is your *AWS account root user*. Its credentials provide complete access to all of your AWS resources.

Note
For security reasons, we recommend that you use the root user only to create an *administrator*, which is an *IAM user* with full permissions to your AWS account. You can then use this administrator to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the *IAM User Guide*.

To create an administrator and sign into the console

1. Create an administrator in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. As an administrator, you can sign in to the console using a special URL. For more information, see How Users Sign in to Your Account in the *IAM User Guide*.

The administrator can create more users in the account. IAM users by default don't have any permissions. The administrator can create users and manage their permissions. For more information, see Creating Your First IAM User and Administrators Group.

For more information about IAM, see the following:

- AWS Identity and Access Management (IAM)
- Getting started
- IAM 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://console.aws.amazon.com/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

You can use the Kinesis Video Streams console, the AWS APIs (CreateSignalingChannel), or the AWS CLI to create your signaling channels.

Create a Signaling Channel Using the Console

1. Sign in to the AWS Management Console and open the Amazon Kinesis Video Streams console at https://console.aws.amazon.com/kinesisvideo/home.
2. On the **Signaling channels** page, choose **Create signaling channel**.
3. On the **Create a new signaling channel** page, type in the name for the signaling channel. Leave the default **Time-to-live (Ttl)** value of 60 seconds unchanged.
4. Choose **Create signaling channel**.

5. After the signaling channel is created, review its details on the channel's details page.

Stream Live Media

The Kinesis Video Streams with WebRTC includes the following SDKs:

- WebRTC SDK in C for Embedded Devices (p. 20)
- WebRTC SDK in JavaScript for Web Applications (p. 22)
- WebRTC SDK for Android (p. 23)
- WebRTC SDK for iOS (p. 26)

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.

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.

Download the Kinesis Video Streams with WebRTC SDK in C

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

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

Build the Kinesis Video Streams with WebRTC SDK in C

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 intaller 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 Samples for the WebRTC SDK in C

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 either of these samples, complete the following steps:

1. Setup your environment with your AWS account credentials:

```
export AWS_ACCESS_KEY_ID= <Your AWS account access Key>
export AWS_SECRET_ACCESS_KEY= <AWS account secret key>
export AWS_KVS_CACERT_PATH= <Full path of your cert.pem file. It is typically available
 in the certs directory inside
Kinesis-video-webrtc-native-build/certs/cert.pm>
```

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 Using the Kinesis Video Streams with WebRTC Test Page (p. 22) 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.

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 at https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-js.

Installing the WebRTC SDK in JavaScript

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

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

The SDK classes are made available in the global window under the `KVSWebRTC` namespace. For example, `window.KVSWebRTC.SignalingClient`.

The SDK is also compatible with bundlers like Webpack. Complete these steps to install the NodeJS module version.

To install the NodeJS module version

1. The preferred way to install the SDK for NodeJS is to use the npm package manager. Run the following command:

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

2. The SDK classes can then be imported like typical NodeJS modules:

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

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

Using the Kinesis Video Streams with WebRTC Test Page

The https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-js GitHub repo hosts a Kinesis Video Streams with WebRTC test page that you can use to create a new or connect to an existing signaling channel and use it as a master or viewer.

The Kinesis Video Streams with WebRTC test page is located at https://awslabs.github.io/amazon-kinesis-video-streams-webrtc-sdk-js/examples/index.html.

1. Open the Kinesis Video Streams with WebRTC test page and specify the following information that you want to use for this demo:

- AWS Region
- The access key and the secret key of the AWS account that you want to use for this demo.
- The name of the signaling channel to which you want to connect.
- Whether you want to send audio, video, or both.

2. If it's a new signaling channel, first choose **Create Channel**. If it's an existing signaling channel, choose either **Start Master** or **Start Viewer** to connect to this channel either as a master or as a viewer.

From there, refer to the example usage in the `examples` directory for how to write an end-to-end WebRTC application that uses the SDK.

Run the WebRTC Test Page Locally

The SDK and test page can be edited and run locally by following these instructions. NodeJS version 8+ is required.

1. Download the WebRTC SDK in JavaScript by running the following command:

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

2. Run `npm install` to download dependencies.
3. Run `npm run develop` to run the webserver.

You have to provide an AWS region, AWS credentials, and a Channel Name to use the WebRTC test page.

The source code for the test page is in the `examples` directory.

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**
> Kinesis Video Streams does not support IPv6 addresses on Android. For more information and steps about disabling IPv6 on your Android device, see https://support.surfshark.com/hc/en-us/articles/360011828279-How-to-disable-IPv6-on-Android-.

Download the WebRTC SDK for Android

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 WebRTC SDK in Android

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 Android WebRTC SDK (p. 25). 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 Android WebRTC SDK (p. 25).
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 Android 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 WebRTC SDK in C for Embedded Devices (p. 20) 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 WebRTC SDK in JavaScript for Web Applications (p. 22) as viewer and verify that the viewer can see the Android master's audio/video.

Configure Amazon Cognito for the Android WebRTC SDK

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services.

Use the following procedures to create a user pool and an identity pool to generate the authentication and authorization settings required for building the Android WebRTC SDK.

To set up Amazon Cognito - User pool

1. Sign in to the Amazon Cognito console.
2. Choose **Manage your User Pools**.
3. Choose **Create a user pool**.
4. Type a value for **Pool name**; for example, `<username>_android_user_pool`.
5. Choose **Review defaults**.
6. Choose **Create pool**.

7. Copy and save the Pool ID value. You will need this value when you configure the identity pool and the `awsconfiguration.json` file in the Android WebRTC SDK.
8. On the page for your pool, choose **App clients**.
9. Choose **Add an app client**.
10. Type a value for **App client name**, for example, `<username>_android_app_client`.
11. Choose **Create app client**.
12. Choose **Show Details**, and copy and save the App client ID and App client secret. You will need these values when you configure the identity pool and the `awsconfiguration.json` file in the Android WebRTC SDK.

To set up Amazon Cognito - Identity pool

1. Open the Amazon Cognito console.
2. Choose **Manage Identity Pools**.
3. Choose **Create new identity pool**.
4. Type a value for Identity pool name, for example, `<username>_android_identity_pool`.
5. Expand the **Authentication providers** section. On the **Cognito** tab, add the values for the `User Pool ID` and `App client ID` from the previous procedure.
6. Choose **Create pool**.
7. On the next page, expand the **Show Details** section.
8. In the section that has a value for Role name that ends in Auth_Role (if you do not have a role, you must create one), choose **View Policy Document**.
9. Choose **Edit**, confirm the **Edit Policy** dialog box that appears, and then paste the following JSON into the editor:

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

10. Choose **Allow**.
11. On the next page, copy and save the Identity pool ID value from the **Get AWS Credentials** code snippet. You need this value to configure the `awsconfiguration.json` file in the Android WebRTC SDK.

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 WebRTC SDK in iOS

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 WebRTC SDK in iOS

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:

   - 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
     ```

   - 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
     ```

   - `Build` again.
3. Configure Amazon Cognito (user pool and identity pool) settings. For details steps, see Configure Amazon Cognito for the iOS WebRTC SDK (p. 29). 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 Android WebRTC SDK (p. 25).

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 Android WebRTC SDK (p. 25).

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 iOS 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 WebRTC SDK in C for Embedded Devices (p. 20) 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 WebRTC SDK in JavaScript for Web Applications (p. 22) as viewer and verify that the JavaScript viewer can see the Android master's audio/video.

Configure Amazon Cognito for the iOS WebRTC SDK

Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps. The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services.

Use the following procedures to create a user pool and an identity pool to generate the authentication and authorization settings required for building the iOS WebRTC SDK.

To set up Amazon Cognito - User pool

1. Sign in to the Amazon Cognito console.

2. Choose **Manage your User Pools**.
3. Choose **Create a user pool**.
4. Type a value for **Pool name**; for example, `<username>_android_user_pool`.
5. Choose **Review defaults**.
6. Choose **Create pool**.
7. Copy and save the Pool ID value. You will need this value when you configure the identity pool and the `awsconfiguration.json` file in the iOS WebRTC SDK.
8. On the page for your pool, choose **App clients**.
9. Choose **Add an app client**.
10. Type a value for **App client name**, for example, `<username>_android_app_client`.
11. Choose **Create app client**.
12. Choose **Show Details**, and copy and save the App client ID and App client secret. You will need these values when you configure the identity pool and the `awsconfiguration.json` file in the iOS WebRTC SDK.

To set up Amazon Cognito - Identity pool

1. Open the Amazon Cognito console.
2. Choose **Manage Identity Pools**.
3. Choose **Create new identity pool**.
4. Type a value for Identity pool name, for example, `<username>_android_identity_pool`.
5. Expand the **Authentication providers** section. On the **Cognito** tab, add the values for the `User Pool ID` and `App client ID` from the previous procedure.
6. Choose **Create pool**.
7. On the next page, expand the **Show Details** section.
8. In the section that has a value for Role name that ends in Auth_Role (if you do not have a role, you must create one), choose **View Policy Document**.
9. Choose **Edit**, confirm the **Edit Policy** dialog box that appears, and then paste the following JSON into the editor:

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

10. Choose **Allow**.
11. On the next page, copy and save the Identity pool ID value from the **Get AWS Credentials** code snippet. You need this value to configure the `awsconfiguration.json` file in the iOS WebRTC SDK.

Client Metrics for the WebRTC 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

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. Calculation is done using Exponential Moving Average (EMA). The associated calls include: describeChannel, createChannel, getChannelEndpoint and deleteChannel.	
dpApiCallLatency	Calculate latency for data plan API calls. Calculation is done using Exponential 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	

Metric	Description	
	most recent uptime value is emitted.	
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.	
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.	

Metric	Description	
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 WebRTC 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 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.	
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. This information can also be extracted from outgoing RTP Stats, but since Ice candidate pair includes a lastPacketSent timestamp, it might be useful to	

Metric	Description	
	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 calculating 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 incomingDataHandler is called. Currently, this also includes the header and padding since the ICE layer is oblivious to the RTP packet format.	
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.	
lastPacketReceivedTimestamp	This is updated on receiving data in `incomingDataHandler()`. This can be used in conjunction with packetsReceived to deduce the current packet receive rate. The start time has to be recorded at the application layer in the `transceiverOnFrame()` callback.	
firstRequestTimestamp	Recorded when the very first STUN binding request is sent out successfully in `iceAgentSendStunPacket()`. This can be used along with lastRequestTimestamp and requestsSent to find average time between STUN binding requests.	
lastRequestTimestamp	Recorded every time a STUN binding request is sent out successfully in `iceAgentSendStunPacket()`.	
lastResponseTimestamp	Recorded every time a STUN binding response is received.	

Metric	Description	
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()`.	
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.	
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.	
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 RtcEncoderStats set by the application.	
framesEncoded	This counter is updated only for video track after successful encoding of the frame. It is	

Metric	Description	
	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.	
framesDiscardedOnSend	This is updated when frame sending fails due to `iceAgentSendPacket` call failure. A frame comprises of a group of packets and currently, framesDiscardedOnSend 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.	
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.	

Metric	Description	
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.	
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.	
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 milliseconds comprising of information such as number of packets sent and bytes sent that	

Metric	Description	
	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/receiver reporfractionLost was sent.	
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 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.	

Metric	Description	
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.	
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_CONNECTING (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, i.e, to	

Metric	Description	
	understand the percentage of bytes that are sent.	
messagesReceived	The metric is incremented in the `onMessage()` callback.	
bytesReceived	The metric is generated in the `onMessage()` callback.	

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 Kinesis Video Streams with WebRTC. The following topics show you how to configure 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 Kinesis Video Streams with WebRTC resources.

Topics

Controlling Access to Kinesis Video Streams with WebRTC Resources Using IAM

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
- IAM User Guide

Contents

- Example Policies for Kinesis Video Streams with WebRTC (p. 47)

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.

Actions for Kinesis Video Streams with WebRTC

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 (*) wildcard 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) for Kinesis Video Streams

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.

Granting 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. The following overview describes the general steps to grant access to channels across accounts:

1. Get the 12-digit account ID of the account that you want to grant permissions to perform operations on your channel (for example, 111111111111).
2. Create a managed policy on the account that owns the channel that allows the level of access that you want to grant. For example policies for Kinesis Video Streams resources, see Example Policies (p. 47) in the next section.
3. Create a role, specifying the account to which you are granting permissions, and attach the policy that you created in the previous step.
4. Create a managed policy that allows the `AssumeRole` action on the role you created in the previous step. For example, the role might look like the following:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::123456789012:role/CustomRole"
  }
}
```

For step-by-step instructions on granting cross-account access, see Delegate Access Across AWS Accounts Using IAM Roles.

Example Policies for Kinesis Video Streams with WebRTC

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.

```
{
    "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.

```
{
    "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.

```
{
```

```
    "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 Kinesis Video Streams with WebRTC

Third-party auditors assess the security and compliance of Kinesis Video Streams with WebRTC as part of multiple AWS compliance programs. These include SOC, PCI, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Kinesis Video Streams with WebRTC is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of Kinesis Video Streams with WebRTC is subject to compliance with standards such as HIPAA or PCI, AWS provides resources to help:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides that might apply to your industry and location
- AWS Config – This AWS service that assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices

Resilience in 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. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

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

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) 1.0. We recommend TLS 1.2 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 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 What Is Least Privilege & Why Do You Need It?

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 (such as a user name and password or access keys) 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 “Logging Kinesis Video Streams with WebRTC API Calls with AWS CloudTrail” (p. 53).

Monitoring

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

Monitoring Kinesis Video Streams with WebRTC Metrics with CloudWatch

You can monitor a Kinesis Video Streams with WebRTC using Amazon CloudWatch, which collects and processes raw data from 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.

Kinesis Video Streams provides the following metrics:

Topics

Signaling Metrics

Metric name	Dimensions	Unit	Description	
Failure	Operation, SignalingChannelName	Count	'0' is emitted if the Operation mentioned in dimension returns 200 status code response. '1' otherwise.	

Metric name	Dimensions	Unit	Description	
Latency	Operation, SignalingChannelName	Milliseconds	The time measured from when the service receives the request until the service returns the response.	
MessagesTransferred	SignalingChannelName	Count	Total number of messages transferred (sent and received) for a given channel.	

The `Operation` dimension can apply to any of the following APIs:

- ConnectAsMaster
- ConnectAsViewer
- SendSdpOffer
- SendSdpAnswer
- SendCandidate
- SendAlexaOfferToMaster
- GetIceServerConfig
- Disconnect

TURN Metrics

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

Logging Kinesis Video Streams with WebRTC 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 Regions. 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: Amazon Kinesis Video Streams with WebRTC 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"
}
```

Document History for the Amazon Kinesis Video Streams with WebRTC Developer Guide

update-history-change	update-history-description	update-history-date
Initial publication	Initial publication of the Amazon Kinesis Video Streams with WebRTC Developer Guide. Learn more	November 4, 2019

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.