KVS Days
Intro to Amazon Kinesis Video Streams WebRTC

Presenter Name(s)
alias@amazon.com

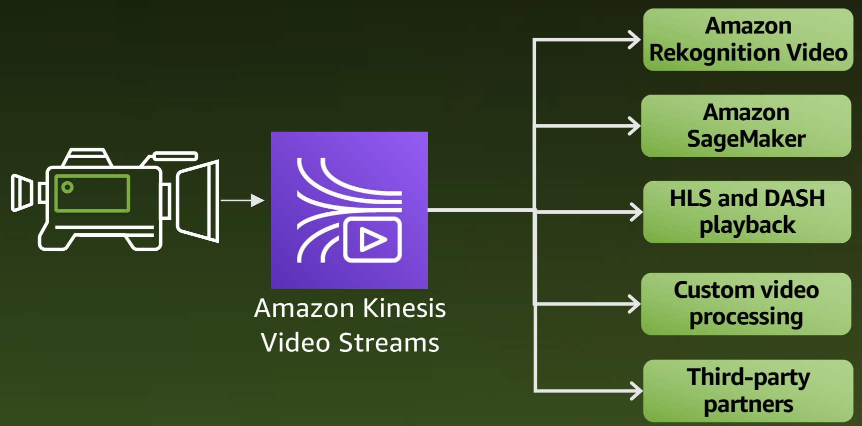
Streaming Options

Kinesis Video Streams

Secure data ingestion from millions of camera devices

Ingest media and store, consume, and play back time-indexed media data

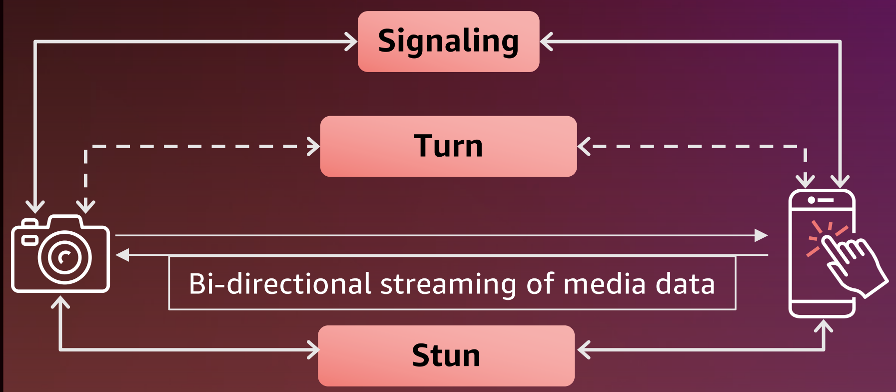
Integration with AI/ML services



Kinesis Video Streams

Low-latency and two-way media streaming with WebRTC

Managed signaling, STUN, and TURN servers



"Stream" and "WebRTC" are used for convenience in this presentation

WebRTC Concepts

WebRTC

Overview

- Web Real-Time Communication (WebRTC)
- An API and a Protocol
- Open Standard
- Collection of existing technologies

Security

- Datagram Transport Layer Security (DTLS)
- Secure Real-time Transport Protocol (SRTP)

Key Concepts

- Signaling, Connecting, Media & Data Comms
- STUN
- TURN
- ICE / Trickle ICE
- SDP
- RTP
- SCTP

WebRTC Concepts

WebRTC

Supported Codecs

- Video: H.264, VP8
- Audio: Opus, G.711 PCMA and PCMU
- Codecs *not* mandated by WebRTC Spec
  - RFC 7742 for video in Browsers w/ WebRTC
  - RFC 7874 for audio in Browsers w/ WebRTC

Network Topologies

- One-to-One
- Full Mesh
- Selective Forwarding Unit (SFU)*
- Multi-point Conferencing Unit (MCU)*

*Currently not supported by KVS WebRTC

WebRTC on Kinesis Video Streams

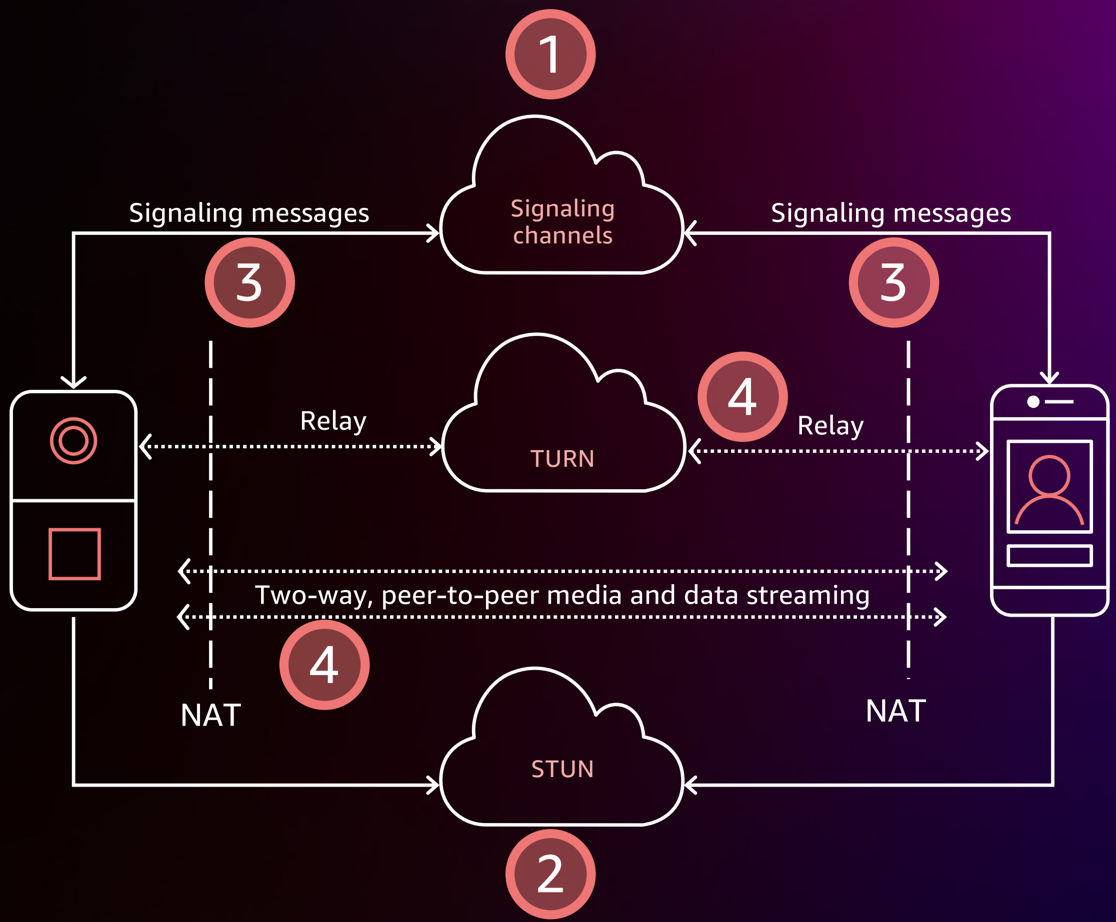
WebRTC

Signaling channels
Kinesis Video Streams cloud resources waiting for devices to connect

STUN
Kinesis Video Streams cloud service enabling the camera and mobile device to discover their public IP addresses and connect to each other peer to peer

Signaling messages
Technical messages sent between the camera and mobile app to establish the live streaming session

TURN
Kinesis Video Streams cloud service relaying media from the camera to mobile app if peer-to-peer connection has failed



WebRTC on Kinesis Video Streams

WebRTC

KVS WebRTC - Key APIs

WebRTC

Control Plane	
CreateSignalingChannel	GetIceServerConfig
DeleteSignalingChannel	SendAlexaOfferToMaster
DescribeSignalingChannel	
GetSignalingChannelEndpoint	
UpdateSignalingChannel	

aws

KVS WebRTC Hostnames, Ports & Protocols

WebRTC

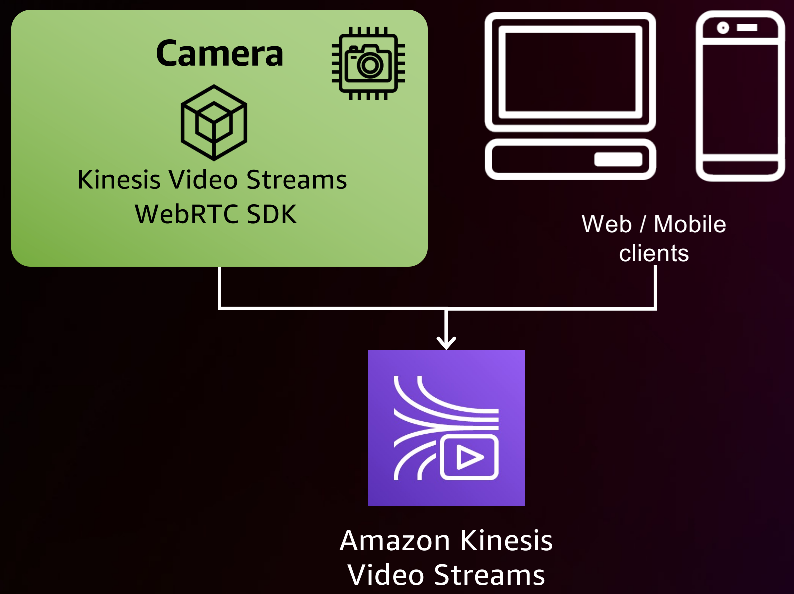
Service	Protocol	Port	Example
Control Plane (GetIceServerConfig)	TCP	443	kinesisvideo.us-west-2.amazonaws.com
HTTPS channel endpoint	TCP	443	r-2c136a55.kinesisvideo.us-west-2.amazonaws.com
WSS channel endpoint	TCP	443	wss://m-26d02974.kinesisvideo.us-west-2.amazonaws.com
STUN endpoint	UDP	443	stun:stun.kinesisvideo.us-west-2.amazonaws.com
TURN endpoint	UDP/TCP	443	turns:34-219-91-62.t-1cd92f6b.kinesisvideo.us-west-2.amazonaws.com:443?transport=udp
ICE Candidate	UDP/TCP	Varies	candidate:0 1 UDP 2122252543 192.168.4.105 59029 typ host candidate:3 1 TCP 2105524479 192.168.4.105 9 typ host tcptype active

WebRTC SDKs

WebRTC

Amazon Kinesis Video Streams WebRTC SDK available in:

- C
- Javascript
- Android
- iOS



GetSignalingChannelEndpoint

WebRTC

- Retrieves endpoints needed to connect to Signaling Channel
- Can be used to retrieve either HTTPS or WSS endpoints
- Must use either the MASTER or VIEWER Role*

```
aws kinesisvideo get-signaling-channel-endpoint \
    --channel-arn $CHANNEL_ARN \
    --single-master-channel-endpoint-configuration Protocols=WSS,HTTPS,Role=MASTER
{
    "ResourceEndpointList": [
        {
            "Protocol": "HTTPS",
            "ResourceEndpoint": "https://[redacted].kinesisvideo.us-east-1.amazonaws.com"
        },
        {
            "Protocol": "WSS",
            "ResourceEndpoint": "wss://[redacted].kinesisvideo.us-east-1.amazonaws.com"
        }
    ]
}
```

*There can only be one MASTER per signaling channel

GetIceServerConfig

WebRTC

- Uses the HTTPS endpoint retrieved from GetSignalingChannelEndpoint
- Use this API to obtain TURN server info (if you want to use TURN)

```
export ICE_SERVER_ENDPOINT_URL=https://[redacted].kinesisvideo.us-east-1.amazonaws.com

aws kinesis-video-signaling get-ice-server-config \
    --channel-arn $CHANNEL_ARN \
    --service TURN \
    --client-id my-amazing-client \
    --username my-turn-user \
    --endpoint-url $ICE_SERVER_ENDPOINT_URL
{
    "IceServerList": [
        {
            "Uris": [
                "turn:[redacted].[redacted].kinesisvideo.us-east-1.amazonaws.com:443?transport=udp",
                "turns:[redacted].[redacted].kinesisvideo.us-east-1.amazonaws.com:443?transport=udp",
                "turns:[redacted].[redacted].kinesisvideo.us-east-1.amazonaws.com:443?transport=tcp"
            ],
            "Username": "[redacted]:[redacted]",
            "Password": "[redacted]",
            "Ttl": 300
        },
        {
            "Uris": [
                "turn:[redacted].[redacted].kinesisvideo.us-east-1.amazonaws.com:443?transport=udp",
                "turns:[redacted].[redacted].kinesisvideo.us-east-1.amazonaws.com:443?transport=udp",
                "turns:[redacted].[redacted].kinesisvideo.us-east-1.amazonaws.com:443?transport=tcp"
            ],
            "Username": "[redacted]:[redacted]",
            "Password": "[redacted]",
            "Ttl": 300
        }
    ]
}
```

Signaling Server

WebRTC

KVS WebRTC Websocket APIs	
ConnectAsMaster	ConnectAsViewer
SendSdpOffer	SendSdpAnswer
SendIceCandidate	Disconnect

- Uses the WSS endpoint returned from GetSignalingChannelEndpoint
- WebSocket APIs
- *WebRTC does not specify transport mechanism for signaling info*

https://docs.aws.amazon.com/kinesisvideostreams-webrtc-dg/latest/devguide/kvswebrtc-websocket-apis.html

Signaling Client

WebRTC

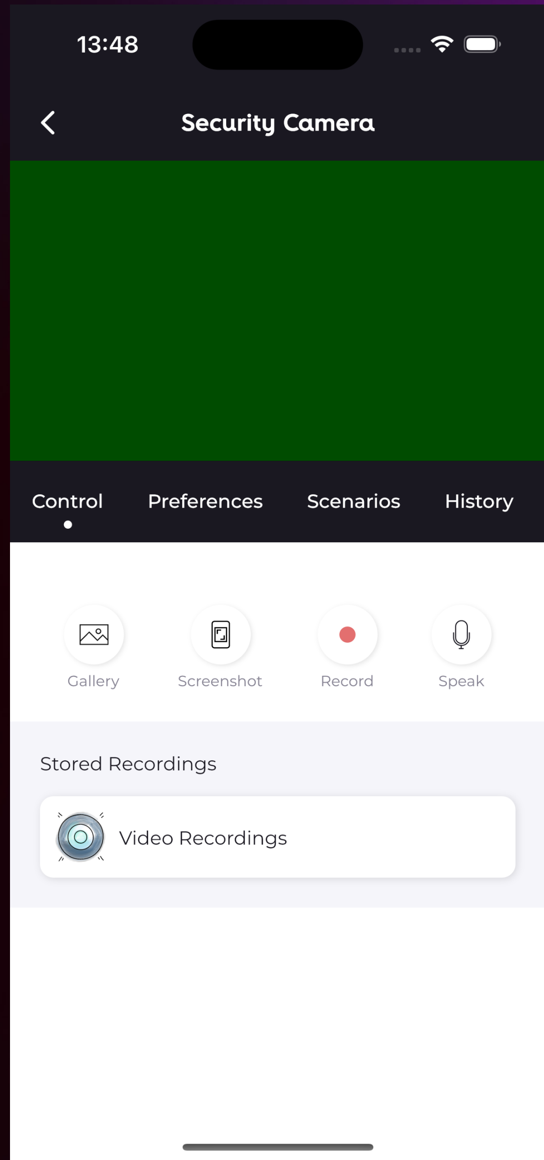
- Kinesis Video Streams WebRTC SDKs provides a Signaling Client
- Implemented in all KVS WebRTC SDKs
- Any open source signaling client can be used
- Remember, the Kinesis Video Streams Signaling Server is implemented as a set of WebSockets APIs

APIs used from a Mobile App

Initialize Live Streaming over WebRTC using the following APIs

1. [DescribeSignalingChannel]
2. [GetSignalingChannelEndpoint]
3. [GetIceServerConfig]
4. Initialize RTCPeerConnection as usual

See the following sample for reference: https://github.com/awslabs/amazon-kinesis-video-streams-webrtc-sdk-js/blob/master/examples/viewer.js#L6



WebRTC

Traversal Using Relays around NAT (TURN)

WebRTC

- Managed component provided by Kinesis Video Streams for WebRTC
- Defined in RFC 8656
- Used when direct, peer-to-peer connectivity cannot be established
  - Could be due to NAT issues between peers, firewalls, proxies, etc
- TURN server *relays* media through a server (instead of peer-to-peer)

Session Traversal Utilities for NAT (STUN)

WebRTC

- Managed component provided by Kinesis Video Streams for WebRTC
- Pre-dates WebRTC; defined in RFC 8489
- Used by both ICE and TURN
- Endpoints are stun:stun.kinesisvideo.{aws-region}.amazonaws.com:443

Interactive Connectivity Establishment (ICE)

WebRTC

- Framework that allows peers to connect with one another
- Pre-dates WebRTC; defined in RFC 8445
- Determines the possible connections available between peers and helps to stay connected
- ICE uses STUN and/or TURN servers to determine ICE candidates comprised of public and/or private IP addresses and port numbers
- ICE candidates can be either UDP or TCP, with UDP being generally preferred
- Client applications can filter out ICE candidates they do not wish to use

Interactive Connectivity Establishment (ICE)

WebRTC

```
candidate:0 1 UDP 2122252543 192.168.4.105 59029 typ host
candidate:3 1 TCP 2105524479 192.168.4.105 9 typ host tcptype active
candidate:3 2 TCP 2105524478 192.168.4.105 9 typ host tcptype active
candidate:0 1 UDP 2122252543 192.168.4.105 60493 typ host
candidate:0 2 UDP 2122252542 192.168.4.105 64240 typ host
candidate:0 2 UDP 2122252542 192.168.4.105 60817 typ host
candidate:3 1 TCP 2105524479 192.168.4.105 9 typ host tcptype active
candidate:3 2 TCP 2105524478 192.168.4.105 9 typ host tcptype active
candidate:4 1 UDP 8331263 107.21.73.246 55396 typ relay raddr 107.21.73.246 rport 55396
candidate:4 2 UDP 8331262 3.238.128.2 63157 typ relay raddr 3.238.128.2 rport 63157
candidate:4 1 UDP 8331263 3.238.128.2 55555 typ relay raddr 3.238.128.2 rport 55555
candidate:4 1 UDP 8331263 3.238.128.2 65250 typ relay raddr 3.238.128.2 rport 65250
candidate:4 1 UDP 8331263 3.238.128.2 54026 typ relay raddr 3.238.128.2 rport 54026
candidate:4 1 UDP 8331263 107.21.73.246 62490 typ relay raddr 107.21.73.246 rport 62490
candidate:4 2 UDP 8331262 107.21.73.246 49663 typ relay raddr 107.21.73.246 rport 49663
candidate:4 2 UDP 8331262 3.238.128.2 50532 typ relay raddr 3.238.128.2 rport 50532
candidate:4 2 UDP 8331262 107.21.73.246 51215 typ relay raddr 107.21.73.246 rport 51215
```

Session Description Protocol (SDP)

WebRTC

- Defined in RFC 8866; Javascript Session Establishment Protocol (JSEP) in RFC 8829
- Key/value protocol
- Not all key values defined in SDP are used in WebRTC
- Contains zero or more Media Descriptions
- Two types: Offers and Answers
- Used to negotiate the connection between peers, determine codecs used, and more

```
v=0
o=- 0 0 IN IP4 127.0.0.1
s=-
c=IN IP4 127.0.0.1
t=0 0
m=audio 4000 RTP/AVP 111
a=rtpmap:111 OPUS/48000/2
m=video 4002 RTP/AVP 96
a=rtpmap:96 VP8/90000
```

WebRTC APIs & Methods

WebRTC

RTCPeerConnection	RTCDataChannel
addTrack() / removeTrack()	close()
addEventListener()	send()
createDataChannel()	
createOffer() / createAnwser()	
getStats()	
restartIce()	
setLocalDescription() / setRemoteDescription()	

WebRTC APIs // Notable Events

WebRTC

RTCPeerConnection	RTCDataChannel
connectionstatechange	close
datachannel	closing
icecandidate	error
iceconnectionstatechange	message
signalingstatechange	open

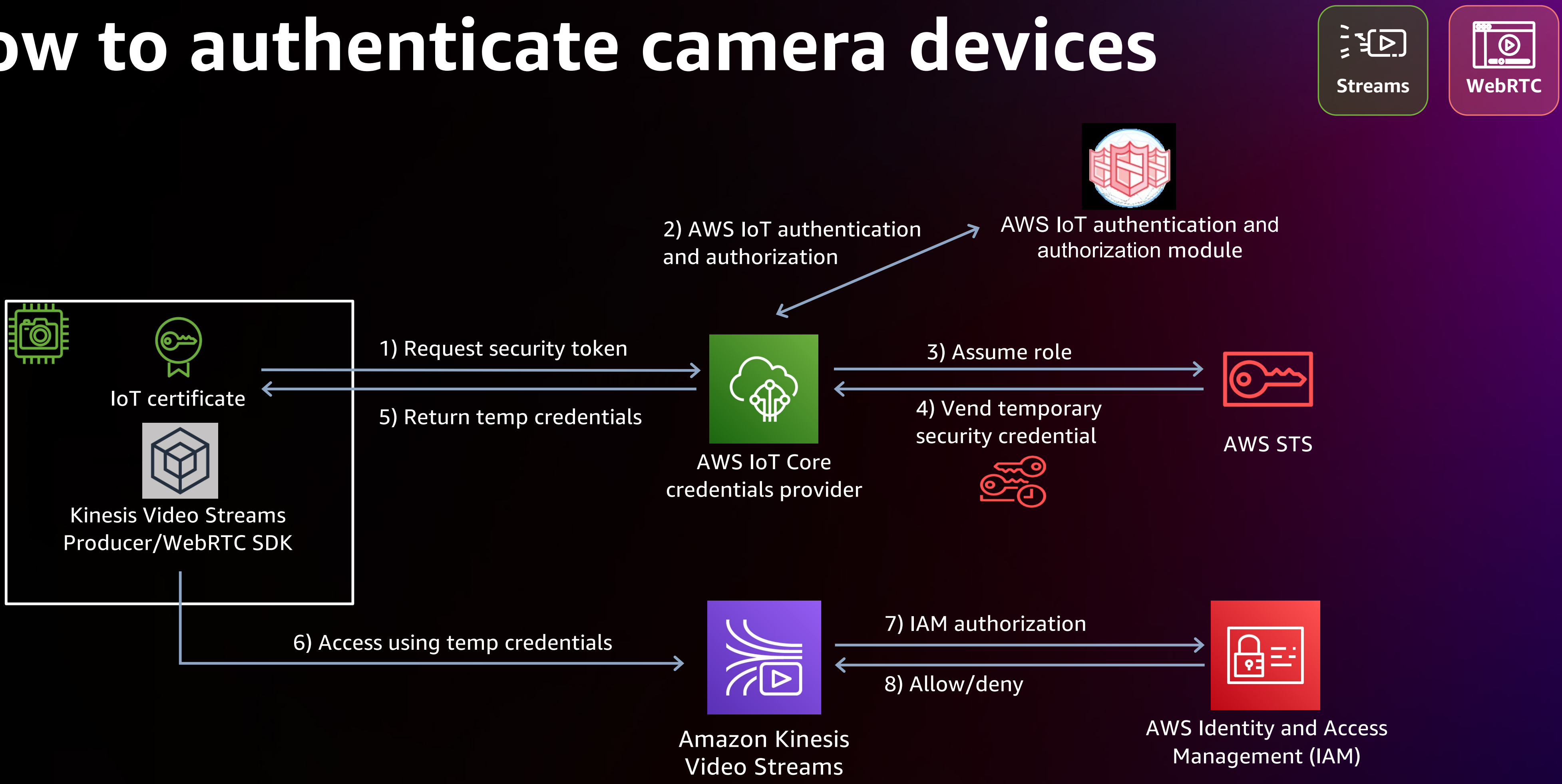
WebRTC APIs + Events // Auto Reconnect

WebRTC

```
//The following will auto reconnect the peer connection in case the network connection is disrupted
viewer.peerConnection.addEventListener('iceconnectionstatechange', async event => {
    console.log('[VIEWER] iceconnectionstatechange event: ' + viewer.peerConnection.iceConnectionState);

    if(viewer.peerConnection.iceConnectionState === 'failed' || viewer.peerConnection.iceConnectionState === 'disconnected'){
        while(viewer.peerConnection.iceConnectionState !== 'connected'){
            console.log(`[VIEWER] iceConnectionState is "${viewer.peerConnection.iceConnectionState}" - attempting to reconnect`)
            viewer.peerConnection.createOffer(
                desc => {
                    console.log(`[VIEWER] Offer from peerconnection:\n${desc.sdp}`);
                    console.log('[VIEWER] peerconnection setLocalDescription start');
                    viewer.peerConnection.setLocalDescription(desc).then(() => (peerconnection) => {
                        console.log(`[VIEWER] peerconnection setRemoteDescription complete`);
                    }, (error) => {
                        console.log(`[VIEWER] Failed to set session description: ${error.toString()}`);
                    })
                },
                error => {
                    console.log(`[VIEWER] Failed to create session description: ${error.toString()}`);
                },
                {
                    iceRestart: true
                }
            );
            await new Promise(r => setTimeout(r, 5000));
        }
    }
});
```

How to authenticate camera devices



https://docs.aws.amazon.com/iot/latest/developerguide/authorizing-direct-aws.html

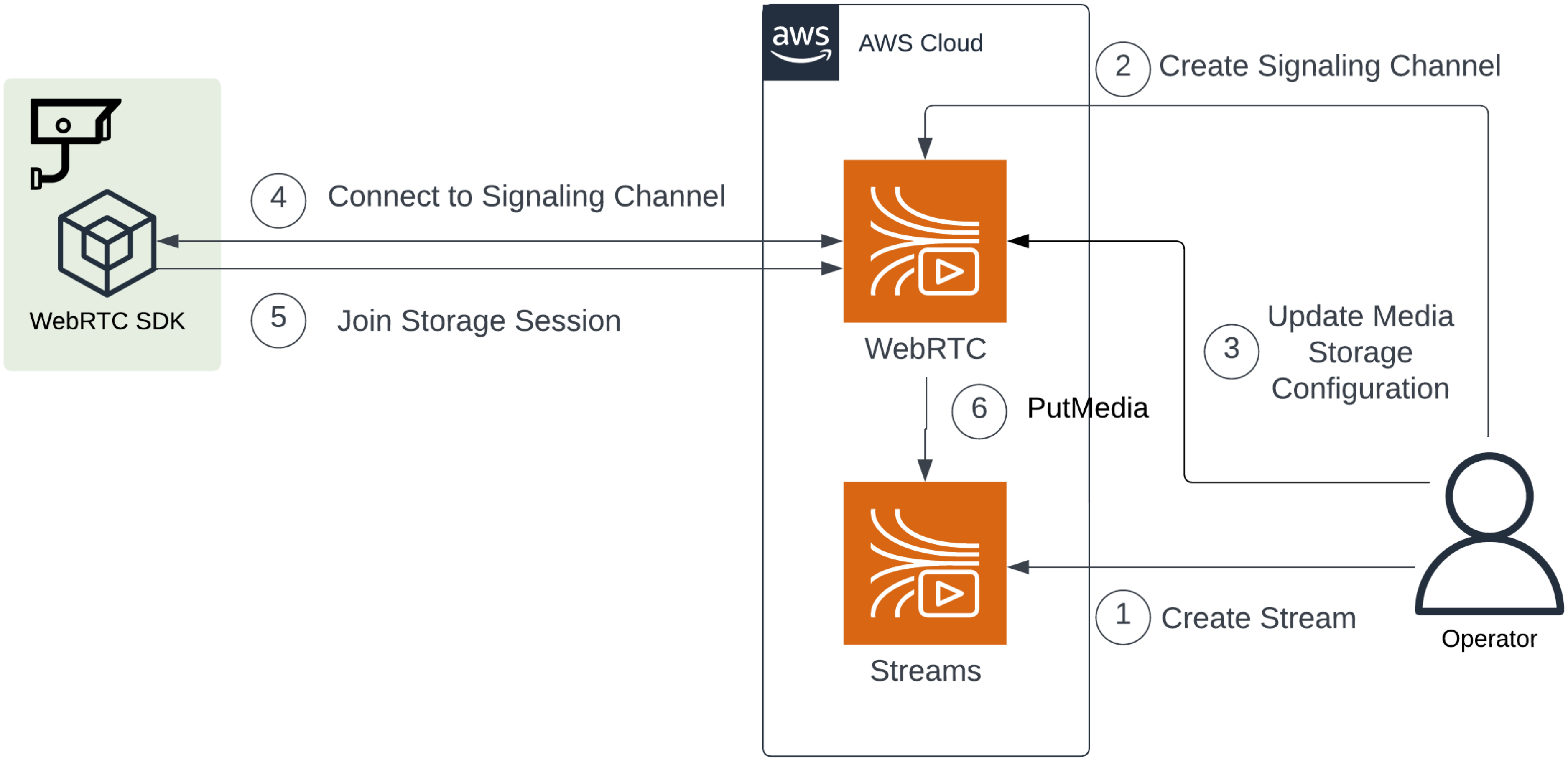
WebRTC Ingest (Preview) - New APIs

WebRTC

- Allows for data to be ingested from the WebRTC SDK to Streams
- During preview, requires separate signaling channel connections for Ingest and for normal Viewers
- Ingested media can be consumed via Streams APIs (HLS/Dash, GetClip, etc)

Control Plane	Data Plane
DescribeMediaStorageConfiguration	JoinStorageSession
UpdateMediaStorageConfiguration	

WebRTC Ingest (Preview)



Service Limits – Control Plane

WebRTC

API	Account service quota: Request	Account service quota: Channels	Channel-level service quota
CreateSignalingChannel	50 TPS [s]	1000 signaling channels per account [s] per region, in all other supported regions.	
DescribeSignalingChannel	300 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	300 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]

Service Limits – Signaling (Websocket) APIs

WebRTC

- 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)
- 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)
- 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)
- GetIceServerConfig
  - API: 5 TPS per signaling channel (hard)

Service Limits - TURN

WebRTC

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

Things we didn't cover

- Bridging other protocols (SIP, etc)
- Handling beyond the initial connection
- Troubleshooting
- getStats reports
- Debugging

A few helpful resources

- https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
- https://www.w3.org/TR/webrtc/
- https://webrtc.github.io/samples/
- https://webrtc.github.io/samples/src/content/peerconnection/bandwidth/
- https://bloggeek.me/trust-webrtc-getstats-accuracy/
- https://bloggeek.me/media-compression-purposefully-losing/
- https://aws.amazon.com/what-is/latency/
- https://testrtc.com/network-jitter-or-round-trip-time-webrtc/
- https://webrtcforthecurious.com/

Thank you!

Please complete the session survey in the **mobile app**