Guía del usuario de transmisión en tiempo real

Amazon IVS

Amazon IVS: Guía del usuario de transmisión en tiempo real

Table of Contents

¿Qué es el streaming en tiempo real de Amazon IVS?

El streaming en tiempo real de Amazon Interactive Video Service (IVS) le ofrece todo lo que necesita para agregar audio y video en tiempo real a sus aplicaciones.

Ventajas:

- Latencia en tiempo real: cree aplicaciones para casos de uso sensibles a la latencia y ayude a sus espectadores a mantenerse conectados e interesados en el streaming en tiempo real de IVS. Ofrezca transmisiones en directo con una latencia inferior a 300 milisegundos desde el host hasta el espectador.
- Alta simultaneidad: libere el potencial de las interacciones a gran escala con el streaming en tiempo real de IVS. Acomode a audiencias de hasta 10 000 espectadores y permita que hasta 12 hosts suban al escenario virtual.
- Optimizado para dispositivos móviles: el streaming en tiempo real de IVS está optimizado para casos de uso móvil y se adapta a una amplia gama de dispositivos y capacidades de red. Al integrar los SDK de transmisión de Amazon IVS para Android y iOS, sus usuarios pueden interactuar como hosts o espectadores y disfrutar de transmisiones en directo de alta calidad en sus dispositivos móviles.

Casos de uso

- Lugares para invitados: cree aplicaciones que permitan a los hosts promocionar a los invitados “al escenario” y convertir a los espectadores en hosts para interactuar en tiempo real.
- Modo Versus (VS): cree experiencias con competiciones paralelas y permita a los espectadores ver a los hosts competir en tiempo real.
- Salas de audio: invite a los oyentes a unirse a la conversación como invitados y fomente una mayor participación en sus salas de audio.
- Subastas de video en vivo: convierta las subastas en eventos de video interactivos y mantenga su entusiasmo e integridad con latencia en tiempo real.

Además de la documentación del producto aquí, consulte https://ivs.rocks/, un sitio dedicado a explorar el contenido publicado (demostraciones, ejemplos de código, publicaciones de blog), calcular el costo y experimentar Amazon IVS a través de demostraciones en vivo.

Solución global, control regional

La transmisión y la visualización son globales

Puede utilizar Amazon IVS para transmitir a espectadores de todo el mundo:

- Cuando realiza una transmisión, Amazon IVS incorpora automáticamente el video en una ubicación cercana a usted.
- Los espectadores pueden ver sus transmisiones en directo en todo el mundo.

Otra forma de decir esto es que el "plano de datos" es global. El plano de datos se refiere al streaming/la incorporación y la visualización.

El control es regional

Aunque el plano de datos de Amazon IVS es global, el "plano de control" es regional. El plano de control hace referencia a la consola, la API y los recursos (escenarios) de Amazon IVS.

Otra forma de decir esto es que Amazon IVS es un “servicio de AWS regional”. Es decir, los recursos de Amazon IVS en cada región son independientes de los recursos similares de otras regiones. Por ejemplo, un escenario que crea en una región es independiente de los escenarios que crea en otras regiones.

Cuando utilice recursos (por ejemplo, creación de un escenario), debe especificar la región en los que se crearán. Posteriormente, al administrar recursos, debe hacerlo desde la misma región en la que se crearon.

Si usa la...	Especifica la región...
Consola de Amazon IVS	Mediante el menú desplegable Select a Region (Seleccionar una región) en la parte superior derecha de la barra de navegación.
API de Amazon IVS	Mediante el punto de enlace de servicio correspondiente. Consulte la Referencia de la API de streaming en tiempo real de Amazon IVS. (Si accede a la API a través de un SDK, configure el parámetro `region` del SDK. Consulte Herramientas para crear en AWS).

Si usa la...	Especifica la región...
AWS CLI	Con cualquiera de las siguientes opciones: • Agregar la `--region <aws-region>` al comando de la CLI. • Colocar la región en el archivo de configuración local de AWS.

Recuerde que, independientemente de la región en la que se haya creado un escenario, puede transmitir a Amazon IVS desde cualquier lugar y los espectadores pueden verlo desde cualquier lugar.

Introducción a transmisión en tiempo real de IVS

Este documento explica los pasos necesarios para integrar la transmisión en tiempo real de Amazon IVS en su aplicación.

Temas

Introducción

Requisitos previos

Antes de usar la transmisión en tiempo real por primera vez, lleve a cabo las siguientes tareas: Para obtener instrucciones, consulte Introducción a transmisión de baja latencia de IVS.

- Cree una cuenta de AWS
- Configuración de usuarios raíz y administrativos

Otras referencias

- Referencia del SDK de transmisión web de IVS
- Referencia del SDK de transmisión para Android de IVS
- Referencia del SDK de transmisión para iOS de IVS
- Referencia de la API de transmisión en tiempo real de IVS

Terminología de transmisión en tiempo real

Plazo	Descripción
Escenario	Un espacio virtual donde los participantes pueden intercambiar videos en tiempo real.
Host	Un participante que envía un video de un entorno local al escenario.
Lector	Un participante que recibe un video de los hosts.
Participante	Un usuario conectado al escenario como host o espectador.
Token de participante	Un token que autentica a un participante cuando se une a un escenario.
SDK de transmisión	Una biblioteca cliente que permite a los participantes enviar y recibir videos.

Información general sobre los pasos

1. the section called “configurar los permisos de IAM”: cree una política de AWS Identity and Access Management (IAM) que proporcione a los usuarios un conjunto básico de permisos y asigne esa política a los usuarios.
2. Creación de un escenario: cree un espacio virtual donde los participantes pueden intercambiar videos en tiempo real.
3. Distribución de los tokens de participante: envíe tokens a los participantes para que puedan unirse a su escenario.

4. Integración del SDK de transmisión de IVS: agregue el SDK de transmisión a su aplicación para permitir a los participantes enviar y recibir videos: the section called “Web”, the section called “Android” y the section called “iOS”.
5. Publicación de videos y suscripción para recibirlos: envíe su video al escenario y reciba videos de otros hosts: the section called “Web”, the section called “Android” ythe section called “iOS”.

configurar los permisos de IAM

A continuación, debe crear una política de AWS Identity and Access Management (IAM) que proporcione a los usuarios una serie de permisos básicos (por ejemplo, para crear un escenario Amazon IVS y crear tokens del participante), y asignar dicha política a los usuarios. Puede asignar los permisos cuando crea un usuario nuevo o agregarlos a un usuario actual. A continuación, se explican ambos procedimientos.

Para obtener más información (por ejemplo, para obtener información sobre los usuarios y las políticas de IAM, cómo adjuntar una política a un usuario y cómo restringir lo que los usuarios pueden hacer con Amazon IVS), consulte:

- Crear un usuario de IAM en la Guía del usuario de IAM.
- La información de Seguridad de Amazon IVS sobre IAM y “Managed Policies for IVS”.
- La información de IAM en Seguridad en Amazon IVS

Puede utilizar una política administrada de AWS existente para Amazon IVS o crear una nueva que personalice los permisos que quiera conceder a un conjunto de usuarios, grupos o roles. A continuación se describen ambos enfoques.

Uso de una política existente para los permisos de IVS

En la mayoría de los casos, querrá utilizar una política administrada de AWS para Amazon IVS. Se describen detalladamente en la sección Managed Policies for IVS de Seguridad de IVS.

- Utilice la política administrada de AWS `IVSReadOnlyAccess` para ofrecer a los desarrolladores de aplicaciones acceso a todos los puntos de conexión de las API Get y List de IVS (para transmisión de baja latencia y en tiempo real).
- Utilice la política administrada de AWS `IVSFullAccess` para ofrecer a los desarrolladores de aplicaciones acceso a todos los puntos de conexión de la API de IVS (para transmisión de baja latencia y en tiempo real).

Opcional: crear una política personalizada para los permisos de Amazon IVS

Siga estos pasos:

1. Inicie sesión en la consola de administración de AWS y abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, seleccione Policies (Políticas) y, a continuación, seleccione Create policy (Crear política). Se abre la ventana Especificar permisos.
3. En la ventana Especificar permisos, elija la pestaña JSON. Luego, copie y pegue la siguiente política de IVS en el área de texto del Editor de políticas. (La política no incluye todas las acciones de Amazon IVS. Puede agregar o eliminar, es decir, permitir o denegar, los permisos de acceso a los puntos de conexión según sea necesario. Consulte la referencia de la API de transmisión en tiempo real de IVS para obtener más información sobre los puntos de conexión de IVS.)

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "ivs:CreateStage",
            "ivs:CreateParticipantToken",
            "ivs:GetStage",
            "ivs:GetStageSession",
            "ivs:ListStages",
            "ivs:ListStageSessions",
            "ivs:CreateEncoderConfiguration",
            "ivs:GetEncoderConfiguration",
            "ivs:ListEncoderConfigurations",
            "ivs:GetComposition",
            "ivs:ListCompositions",
            "ivs:StartComposition",
            "ivs:StopComposition"
          ],
          "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:DescribeAlarms",
```

```
            "cloudwatch:GetMetricData",
            "s3:DeleteBucketPolicy",
            "s3:GetBucketLocation",
            "s3:GetBucketPolicy",
            "s3:PutBucketPolicy",
            "servicequotas:ListAWSDefaultServiceQuotas",
            "servicequotas:ListRequestedServiceQuotaChangeHistoryByQuota",
            "servicequotas:ListServiceQuotas",
            "servicequotas:ListServices",
            "servicequotas:ListTagsForResource"
        ],
        "Resource": "*"
    }
  ]
}
```

4. En la ventana Especificar permisos, elija Siguiente (desplácese a la parte inferior de la ventana para verlo). Se abre la ventana Revisar y crear.
5. En la ventana Revisar y crear, asígnele un nombre a la política y, si lo desea, agregue una descripción. Anote el nombre de la política, ya que lo necesitará cuando cree usuarios (más adelante). Elija Create policy (Crear política) (en la parte inferior de la ventana).
6. Volverá a la ventana de la consola de IAM, donde debería ver un banner que confirma la creación de la política nueva.

Crear usuarios nuevos y agregar permisos

Claves de acceso de usuario de IAM

Las claves de acceso de IAM constan de un ID de clave de acceso y de una clave de acceso secreta. Se utilizan para firmar las solicitudes programáticas que realiza a AWS. Si no tiene claves de acceso, puede crearlas mediante la consola de administración de AWS. Como práctica recomendada, no cree claves de acceso del usuario raíz.

El único momento que puede ver o descargar la clave de acceso secreta es cuando crea las claves de acceso. No puede recuperarla más adelante. Sin embargo, puede crear claves de acceso nuevas en cualquier momento; debe tener permisos para realizar las acciones de IAM requeridas.

Siempre almacene las claves de acceso de forma segura. Nunca las comparta con terceros (incluso si parece que la consulta proviene de Amazon). Para obtener más información, consulte Administración de claves de acceso para usuarios de IAM en la Guía del usuario de IAM de .

Procedimiento

Siga estos pasos:

1. En el panel de navegación, elija Usuarios y la opción Agregar usuario. Se abre la ventana Especificar los detalles del usuario.
2. En la ventana Especificar los detalles del usuario:
   a. En Detalles del usuario, escriba el nuevo nombre de usuario que se va a crear.
   b. Active la casilla de verificación Acceso de usuario a la consola de administración de AWS.
   c. En Contraseña de la consola, seleccione Contraseña generada de manera automática.
   d. Seleccione la casilla de verificación El usuario debe crear una contraseña nueva en el siguiente inicio de sesión.
   e. Elija Siguiente. Se abre la ventana Establecer permisos.
3. En Establecer permisos, elija Asociar directamente las políticas existentes. Se abre la ventana Políticas de permisos.
4. En el cuadro de búsqueda, ingrese el nombre de una política de IVS (ya sea una política administrada de AWS o una política personalizada creada con anterioridad). Cuando la encuentre, marque la casilla para seleccionar la política.
5. Elija Siguiente (en la parte inferior de la ventana). Se abre la ventana Revisar y crear.
6. En la ventana Revisar y crear, confirme que toda la información del usuario sea correcta y, a continuación, elija Crear usuario (en la parte inferior de la ventana).
7. Se abre la ventana Recuperar la contraseña, que contiene los detalles de inicio de sesión de la consola. Guarde esta información de forma segura para consultarla en el futuro. Cuando haya terminado, elija Volver a la lista de usuarios.

Agregar permisos para un usuario existente

Siga estos pasos:

1. Inicie sesión en la consola de administración de AWS y abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación, elija Users (Usuarios) y, a continuación, elija un nombre de usuario existente para actualizarlo. (Haga clic en el nombre para elegirlo; no marque la casilla de selección.)

3. En la página Resumen, en la pestaña Permisos, elija Agregar permisos. Se abre la ventana Agregar permisos.
4. Seleccione Asociar directamente las políticas existentes. Se abre la ventana Políticas de permisos.
5. En el cuadro de búsqueda, ingrese el nombre de una política de IVS (ya sea una política administrada de AWS o una política personalizada creada con anterioridad). Cuando encuentre la política, marque la casilla para seleccionarla.
6. Elija Siguiente (en la parte inferior de la ventana). Se abre la ventana Revisión.
7. En la ventana de Revisión, selecciona Agregar permisos (en la parte inferior de la ventana).
8. En la página Summary (Resumen), confirme que se agregó la política de IVS.

Creación de un escenario

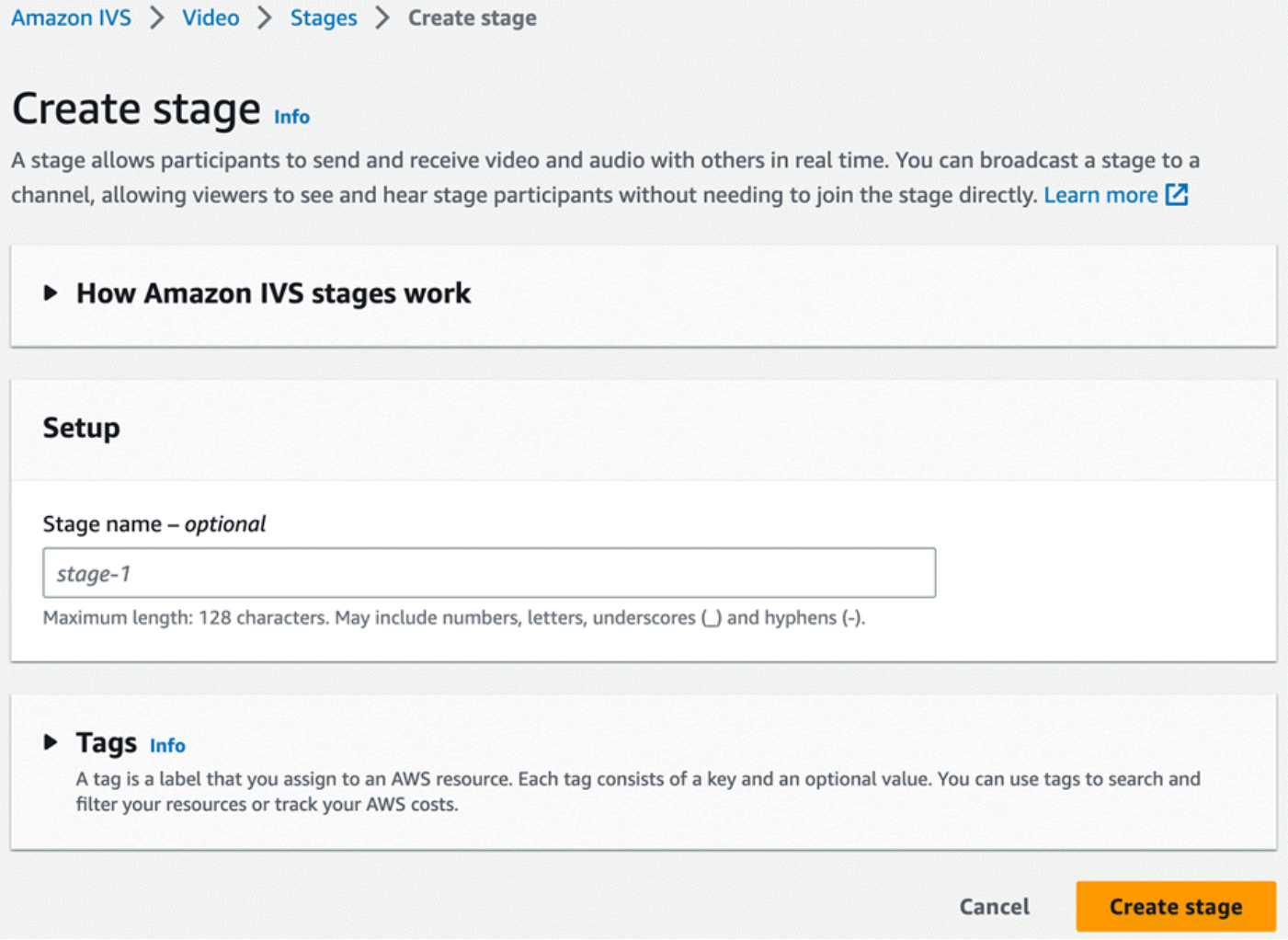
Un escenario es un espacio virtual donde los participantes pueden intercambiar videos en tiempo real. Es el recurso fundamental de la API de transmisión en tiempo real. Puede crear un escenario mediante la consola o el CreateStage punto final.

Le recomendamos que, siempre que sea posible, cree un nuevo escenario para cada sesión lógica y la elimine cuando termine, en lugar de conservar los escenarios antiguos para su posible reutilización. Si los recursos obsoletos (escenarios antiguos que no se van a reutilizar) no se eliminan, es probable que alcance el límite de escenarios más rápido.

Instrucciones de la consola

1. Abra la consola de Amazon IVS.

   (También puede acceder a la consola de Amazon IVS a través de la consola de administración de AWS).
2. En el panel de navegación de la izquierda, selecciona Fases y, a continuación, seleccione Crear fase. Aparece la ventana Crear fase.



3. Si lo desea, ingrese un nombre para la fase. Seleccione Crear fase para crear la fase. Aparece la página de detalles de la nueva fase.

Instrucciones de la CLI

Para instalar la AWS CLI, consulte Instalación o actualización de la versión más reciente de AWS CLI.

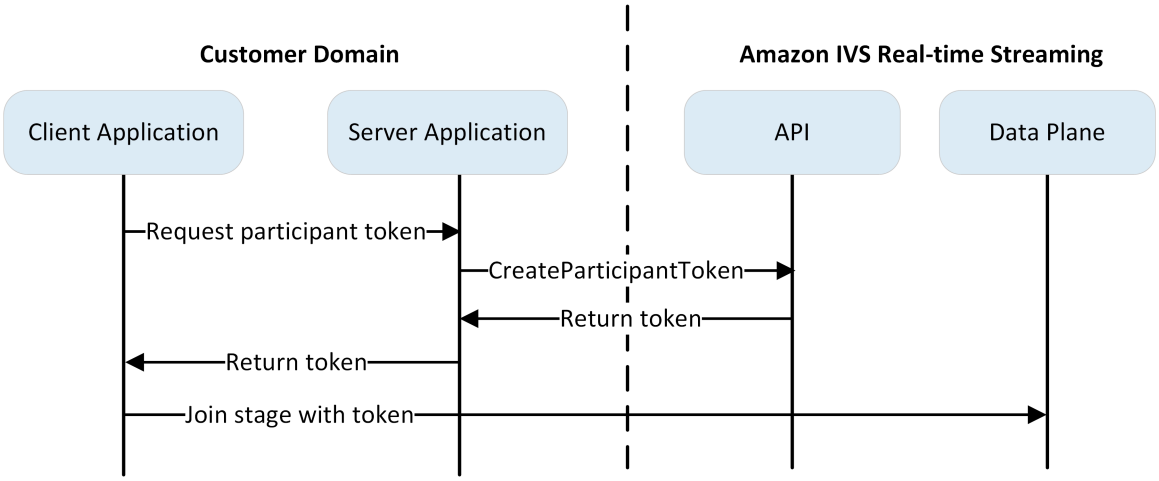
Ahora puede utilizar la CLI para crear y administrar recursos. La API de escenarios se encuentra en el espacio de nombres ivs-realtime. Por ejemplo, para crear un escenario:

```
aws ivs-realtime create-stage --name "test-stage"
```

La respuesta es:

```
{
   "stage": {
      "arn": "arn:aws:ivs:us-west-2:376666121854:stage/VSWjvX5XOkU3",
      "name": "test-stage"
   }
}
```

Distribución de los tokens de participante



Ahora que tiene un escenario, debe crear y distribuir tokens a los participantes para que puedan unirse al escenario y empezar a enviar y recibir videos.

Como se muestra arriba, una aplicación cliente solicita un token a la aplicación de servidor y la aplicación de servidor llama CreateParticipantToken mediante una solicitud firmada por el AWS SDK o SigV4. Como las credenciales de AWS se utilizan para llamar a la API, el token debe generarse en una aplicación segura del lado del servidor, no en la aplicación del lado del cliente.

Al crear un token de participante, puede especificar opcionalmente las capacidades que habilita ese token. El valor predeterminado es `PUBLISH` y `SUBSCRIBE`, que permite al participante enviar y recibir audio y video, pero puede emitir tokens con un subconjunto de capacidades. Por ejemplo, puede emitir un token que tenga únicamente la capacidad `SUBSCRIBE` para los moderadores. En ese caso, los moderadores podrían ver a los participantes que envían videos, pero no enviar los suyos propios.

Puede crear tokens de participantes mediante la consola o la CLI para llevar a cabo pruebas y desarrollar, pero lo más probable es que le convenga más crearlos con el SDK de AWS en su entorno de producción.

Tendrá que distribuir los tokens de su servidor a cada cliente (por ejemplo, mediante una solicitud a la API). No ofrecemos esta funcionalidad. Para esta guía, basta con seguir estos pasos para copiar y pegar los tokens en el código del cliente:

Importante: trate los tokens como elementos opacos; es decir, no cree funciones en función del contenido del token. El formato de los tokens podría cambiar en el futuro.

Instrucciones de la consola

1. Vaya al escenario que creó en el paso anterior.
2. Seleccione Crear un token de participante. Verá la ventana Crear un token de participante.
3. Ingrese un ID de usuario para asociarlo al token. Puede ser cualquier texto codificado en UTF-8.
4. Seleccione Crear un token de participante.
5. Copie el token. Importante: Asegúrese de guardar el token; IVS no lo almacena y no podrá recuperarlo más adelante.

Instrucciones de la CLI

Crear un token con la AWS CLI requiere que antes descargue y configure la CLI en su equipo. Para obtener más información, consulte la Guía del usuario de la interfaz de línea de comandos de AWS. Nota: La generación de tokens con la AWS CLI es una buena opción para hacer pruebas, pero, para el uso en producción, le recomendamos que genere tokens en el lado del servidor con el SDK de AWS (consulte las instrucciones a continuación).

1. Ejecute el comando `create-participant-token` con el ARN del escenario. Incluya cualquiera de las siguientes capacidades: `"PUBLISH"`, `"SUBSCRIBE"`.

```
aws ivs-realtime create-participant-token --stage-arn arn:aws:ivs:us-
west-2:376666121854:stage/VSWjvX5XOkU3 --capabilities '["PUBLISH", "SUBSCRIBE"]'
```

2. Esto devuelve un token de participante:

```
{
    "participantToken": {
        "capabilities": [
            "PUBLISH",
            "SUBSCRIBE"
        ],
```

```
        "expirationTime": "2023-06-03T07:04:31+00:00",
        "participantId": "tU06DT5jCJeb",
        "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtdmVFTm9sUyIsInJ
TaKjllW9Qac6c5xBrdAk"    }
}
```

3. Guarde este token. Lo necesitará para unirse al escenario y enviar y recibir videos.

Instrucciones del SDK de AWS

Puede utilizar el SDK de AWS para crear tokens. A continuación se muestran las instrucciones de uso del AWS SDK JavaScript.

Importante: Este código debe ejecutarse en el lado del servidor y su salida se debe pasar al cliente.

Requisito previo: para usar el ejemplo de código que aparece a continuación, debe instalar el paquete aws-sdk/client-ivs-realtime . Para obtener más información, consulte Introducción al AWS SDK para JavaScript.

```
import { IVSRealTimeClient, CreateParticipantTokenCommand } from "@aws-sdk/client-ivs-
realtime";

const ivsRealtimeClient = new IVSRealTimeClient({ region: 'us-west-2' });
const stageArn = 'arn:aws:ivs:us-west-2:123456789012:stage/L210UYabcdef';
const createStageTokenRequest = new CreateParticipantTokenCommand({
  stageArn,
});
const response = await ivsRealtimeClient.send(createStageTokenRequest);
console.log('token', response.participantToken.token);
```

Integración del SDK de transmisión de IVS

IVS proporciona un SDK de transmisión para Android, iOS y web que puede integrar en su aplicación. El SDK de transmisión se utiliza tanto para enviar como para recibir videos. En esta sección, escribimos una aplicación sencilla que permite que dos o más participantes interactúen en tiempo real. Los pasos que se indican a continuación le guiarán por el proceso de creación de una aplicación llamada BasicRealTime. El código completo de la aplicación está activado CodePen y GitHub:

- Web: https://codepen.io/amazon-ivs/pen/ZEqgrpo/cbe7ac3b0ecc8c0f0a5c0dc9d6d36433
- Android: https://github.com/aws-samples/ amazon-ivs-real-time - streaming-android-samples
- iOS: https://github.com/aws-samples/ amazon-ivs-real-time - streaming-ios-samples

Web

Configuración de archivos

Para empezar, cree una carpeta y un archivo HTML y JS inicial para configurar los archivos:

```
mkdir realtime-web-example
cd realtime-web-example
touch index.html
touch app.js
```

Puede instalar el SDK de transmisión mediante una etiqueta script o npm. Nuestro ejemplo usa la etiqueta script por motivos de simplicidad, pero es fácil de modificar si opta por usar npm más adelante.

Uso de una etiqueta de script

El SDK para retransmisiones web se distribuye como una JavaScript biblioteca y se puede consultar en https://web-broadcast.live-video.net/1.8.0/ amazon-ivs-web-broadcast .js.

Cuando se carga mediante una etiqueta `<script>`, la biblioteca muestra una variable global en el ámbito del intervalo denominado `IVSBroadcastClient`.

Con npm

Para instalar el paquete de npm:

```
npm install amazon-ivs-web-broadcast
```

Ahora puede acceder al objeto IVS: BroadcastClient

```
const { Stage } = IVSBroadcastClient;
```

Android

Creación del proyecto para Android

1. Cree un nuevo proyecto en Android Studio.
2. Elija Actividad de vistas vacías.

   Nota: En algunas versiones anteriores de Android Studio, la actividad basada en vistas se denomina Actividad vacía. Si aparece la ventana de Android Studio Actividad vacía y no Actividad de vistas vacías, seleccione Actividad vacía. De lo contrario, no seleccione Actividad vacía, ya que utilizaremos las API de vistas (no Jetpack Compose).
3. Asigne un nombre a su proyecto y, a continuación, seleccione Finalizar.

Instalación del SDK de transmisión

A fin de agregar la biblioteca de transmisión de Android de Amazon IVS a su entorno de desarrollo de Android, agregue la biblioteca al archivo `build.gradle` del módulo como se muestra a continuación (para la versión más reciente del SDK de transmisión de Amazon IVS). En los proyectos más recientes, es posible que el repositorio `mavenCentral` ya esté incluido en su archivo `settings.gradle`, si ese es el caso, puede omitir el bloqueo de `repositories`. Para nuestro ejemplo, también necesitaremos habilitar el enlace de datos en el bloque `android`.

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
}

dependencies {
     implementation 'com.amazonaws:ivs-broadcast:1.14.1:stages@aar'
}
```

Alternativamente, para instalar el SDK de forma manual, descargue la última versión desde esta ubicación:

https://search.maven.org/artifact/com.amazonaws/ivs-broadcast

iOS

Creación del proyecto para iOS

1. Cree un nuevo proyecto de Xcode.
2. En Plataforma, seleccione iOS.
3. En Aplicación, seleccione Aplicación.
4. Ingrese el nombre del producto de su aplicación y, a continuación, seleccione Siguiente.
5. Para elegir un directorio en el que guardar el proyecto, vaya a este y, a continuación, seleccione Crear.

A continuación, tiene que incorporar el SDK. Le recomendamos que integre el SDK de transmisión mediante CocoaPods. También puede agregar el marco a su proyecto de forma manual. Ambos métodos se describen a continuación.

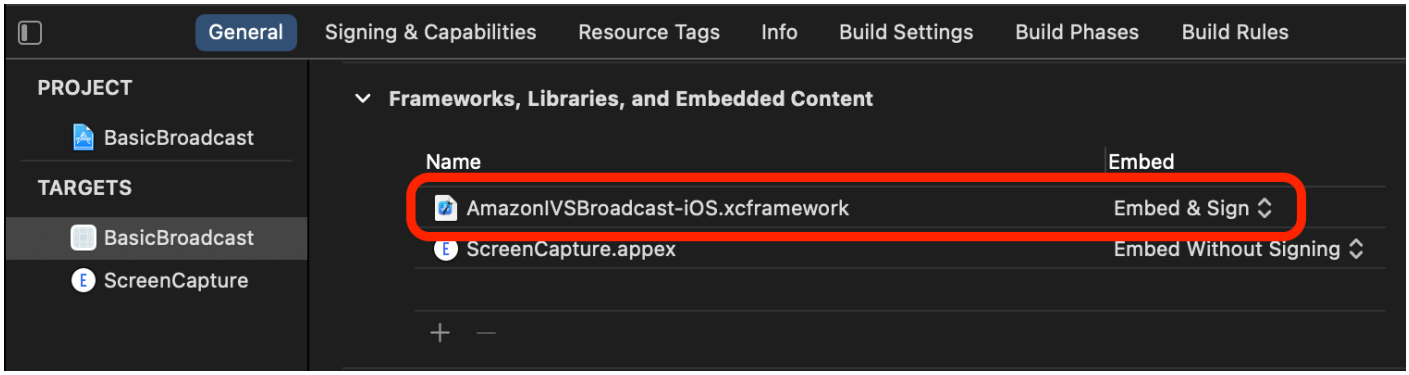
Recomendado: instale el SDK de transmisión (CocoaPods)

Supongamos que el nombre de su proyecto es `BasicRealTime`; cree un `Podfile` en la carpeta del proyecto con el siguiente contenido y, a continuación, ejecute `pod install`:

```
target 'BasicRealTime' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for BasicRealTime
  pod 'AmazonIVSBroadcast/Stages'
end
```

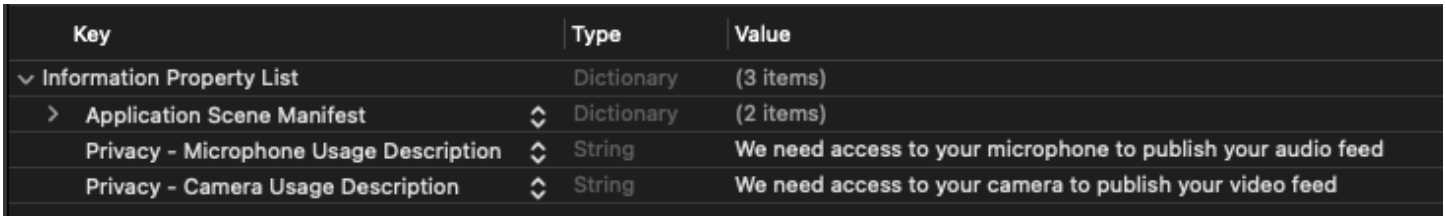
Método alternativo: instalar el marco de forma manual

1. Descargue la última versión desde https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Extraiga el contenido del archivo. `AmazonIVSBroadcast.xcframework` contiene el SDK para el dispositivo y el simulador.
3. Integre el `AmazonIVSBroadcast.xcframework` arrastrándolo a la sección Marcos, librerías y contenido integrado de la pestaña General para el destino de la aplicación:



Configuración de permisos

Tiene que actualizar el `Info.plist` de su proyecto para agregar dos entradas nuevas para `NSCameraUsageDescription` y `NSMicrophoneUsageDescription`. En los valores, proporcione explicaciones para el usuario sobre por qué la aplicación solicita acceso a la cámara y al micrófono.



Publicación de videos y suscripción para recibirlos

Consulta los detalles a continuación para la web, Android e iOS.

Web

Creación de una plantilla HTML reutilizable

Primero, creemos la plantilla HTML reutilizable e importemos la biblioteca como una etiqueta script:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
  <!-- Import the SDK -->
  <script src="https://web-broadcast.live-video.net/1.8.0/amazon-ivs-web-
broadcast.js"></script>
</head>

<body>

<!-- TODO - fill in with next sections -->
<script src="./app.js"></script>

</body>
</html>
```

Configuración de la entrada de tokens y adición de botones de unión y salida

Aquí completamos el cuerpo con nuestros controles de entrada. Estos toman como entrada el token y configuran botones de unión y salida. Por lo general, las aplicaciones solicitarán el token a la API de la aplicación, pero, en este ejemplo, tiene que copiar y pegar el token en la entrada del token.

```
<h1>IVS Real-Time Streaming</h1>
<hr />

<label for="token">Token</label>
<input type="text" id="token" name="token" />
<button class="button" id="join-button">Join</button>
<button class="button" id="leave-button" style="display: none;">Leave</button>
<hr />
```

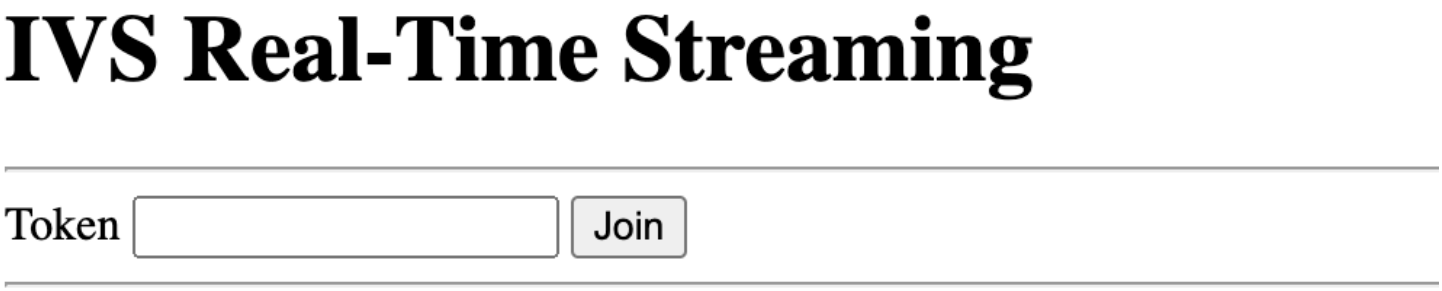
Adición de elementos de contenedor multimedia

Estos elementos albergarán el contenido multimedia para nuestros participantes locales y remotos. Agregamos una etiqueta script para cargar la lógica de nuestra aplicación definida en `app.js`.

```
<!-- Local Participant -->
<div id="local-media"></div>

<!-- Remote Participants -->
<div id="remote-media"></div>

<!-- Load Script -->
```

```
<script src="./app.js"></script>
```

Esto completa la página HTML y debería ver esto al cargar `index.html` en un navegador:



Creación de app.js

Pasemos a definir el contenido de nuestro archivo `app.js`. Comience por importar todas las propiedades necesarias de la versión global del SDK:

```
const {
  Stage,
  LocalStageStream,
  SubscribeType,
  StageEvents,
  ConnectionState,
  StreamType
} = IVSBroadcastClient;
```

Creación de las variables de la aplicación

Configure variables que contengan referencias a nuestros botones HTML de unión y salida y al estado de la aplicación:

```
let joinButton = document.getElementById("join-button");
let leaveButton = document.getElementById("leave-button");

// Stage management
let stage;
let joining = false;
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
let micStageStream;
```

Creación de joinStage 1: definición de la función y validación de la entrada

La función `joinStage` toma el token de entrada, crea una conexión con el escenario y comienza a publicar el video y el audio recuperados de `getUserMedia`.

Para empezar, definimos la función y validamos el estado y la entrada del token. Vamos a desarrollar esta función en las próximas secciones.

```
const joinStage = async () => {
  if (connected || joining) {
    return;
  }
  joining = true;

  const token = document.getElementById("token").value;

  if (!token) {
    window.alert("Please enter a participant token");
    joining = false;
    return;
  }

  // Fill in with the next sections
};
```

Creación de joinStage 2: obtención de contenido multimedia para su publicación

Este es el contenido multimedia que se publicará en el escenario:

```
async function getCamera() {
  // Use Max Width and Height
  return navigator.mediaDevices.getUserMedia({
    video: true,
    audio: false
  });
}

async function getMic() {
  return navigator.mediaDevices.getUserMedia({
    video: false,
    audio: true
  });
```

```
}

// Retrieve the User Media currently set on the page
localCamera = await getCamera();
localMic = await getMic();

// Create StageStreams for Audio and Video
cameraStageStream = new LocalStageStream(localCamera.getVideoTracks()[0]);
micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);
```

Creación de joinStage 3: definición de la estrategia escénica y creación del escenario

Esta estrategia de escenarios es la base de la lógica de decisión que utiliza el SDK para decidir qué publicar y a qué participantes suscribirse. Para obtener más información sobre el propósito de la función, consulte Estrategia.

Esta estrategia es sencilla. Cuando se una al escenario, publique las transmisiones que acabamos de recuperar y suscríbase al contenido de audio y video de todos los participantes remotos:

```
const strategy = {
  stageStreamsToPublish() {
    return [cameraStageStream, micStageStream];
  },
  shouldPublishParticipant() {
    return true;
  },
  shouldSubscribeToParticipant() {
    return SubscribeType.AUDIO_VIDEO;
  }
};

stage = new Stage(token, strategy);
```

Creación de joinStage 4: gestión de los eventos del escenario y renderización del contenido multimedia

Los escenarios emiten muchos eventos. Tendremos que escuchar los eventos `STAGE_PARTICIPANT_STREAMS_ADDED` y `STAGE_PARTICIPANT_LEFT` para renderizar y eliminar contenido multimedia hacia y desde la página. Puede encontrar un conjunto más exhaustivo de eventos en Eventos.

Tenga en cuenta que creamos cuatro funciones auxiliares para poder administrar los elementos del DOM
necesarios: `setupParticipant`, `teardownParticipant`, `createVideoEl` y `createContainer`.

```
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
  connected = state === ConnectionState.CONNECTED;

  if (connected) {
    joining = false;
    joinButton.style = "display: none";
    leaveButton.style = "display: inline-block";
  }
});

stage.on(
  StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED,
  (participant, streams) => {
    console.log("Participant Media Added: ", participant, streams);

    let streamsToDisplay = streams;

    if (participant.isLocal) {
      // Ensure to exclude local audio streams, otherwise echo will occur
      streamsToDisplay = streams.filter(
        (stream) => stream.streamType === StreamType.VIDEO
      );
    }

    const videoEl = setupParticipant(participant);
    streamsToDisplay.forEach((stream) =>
      videoEl.srcObject.addTrack(stream.mediaStreamTrack)
    );
  }
);

stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {
  console.log("Participant Left: ", participant);
  teardownParticipant(participant);
});


// Helper functions for managing DOM
```

```
function setupParticipant({ isLocal, id }) {
  const groupId = isLocal ? "local-media" : "remote-media";
  const groupContainer = document.getElementById(groupId);

  const participantContainerId = isLocal ? "local" : id;
  const participantContainer = createContainer(participantContainerId);
  const videoEl = createVideoEl(participantContainerId);

  participantContainer.appendChild(videoEl);
  groupContainer.appendChild(participantContainer);

  return videoEl;
}

function teardownParticipant({ isLocal, id }) {
  const groupId = isLocal ? "local-media" : "remote-media";
  const groupContainer = document.getElementById(groupId);
  const participantContainerId = isLocal ? "local" : id;

  const participantDiv = document.getElementById(
    participantContainerId + "-container"
  );
  if (!participantDiv) {
    return;
  }
  groupContainer.removeChild(participantDiv);
}

function createVideoEl(id) {
  const videoEl = document.createElement("video");
  videoEl.id = id;
  videoEl.autoplay = true;
  videoEl.playsInline = true;
  videoEl.srcObject = new MediaStream();
  return videoEl;
}

function createContainer(id) {
  const participantContainer = document.createElement("div");
  participantContainer.classList = "participant-container";
  participantContainer.id = id + "-container";

  return participantContainer;
```

```
}
```

Creación de joinStage 5: unión al escenario

Vamos a unirnos por fin al escenario para completar la función `joinStage`.

```
try {
  await stage.join();
} catch (err) {
  joining = false;
  connected = false;
  console.error(err.message);
}
```

Creación de leaveStage

Defina la función `leaveStage` que invocará el botón de salida.

```
const leaveStage = async () => {
  stage.leave();

  joining = false;
  connected = false;
};
```

Inicialización de los controladores de eventos de entrada

Agregaremos una última función al archivo `app.js`. Esta función se invoca inmediatamente cuando se carga la página y establece controladores de eventos para unirse al escenario y salir de este.

```
const init = async () => {
  try {
    // Prevents issues on Safari/FF so devices are not blank
    await navigator.mediaDevices.getUserMedia({ video: true, audio: true });
  } catch (e) {
    alert(
      "Problem retrieving media! Enable camera and microphone permissions."
    );
  }
```

```
  joinButton.addEventListener("click", () => {
    joinStage();
  });

  leaveButton.addEventListener("click", () => {
    leaveStage();
    joinButton.style = "display: inline-block";
    leaveButton.style = "display: none";
  });
};

init(); // call the function
```

Ejecute la aplicación y proporcione un token

En este punto, puede compartir la página web de forma local o con otras personas, abrir la página, introducir un token de participante y unirte al escenario.

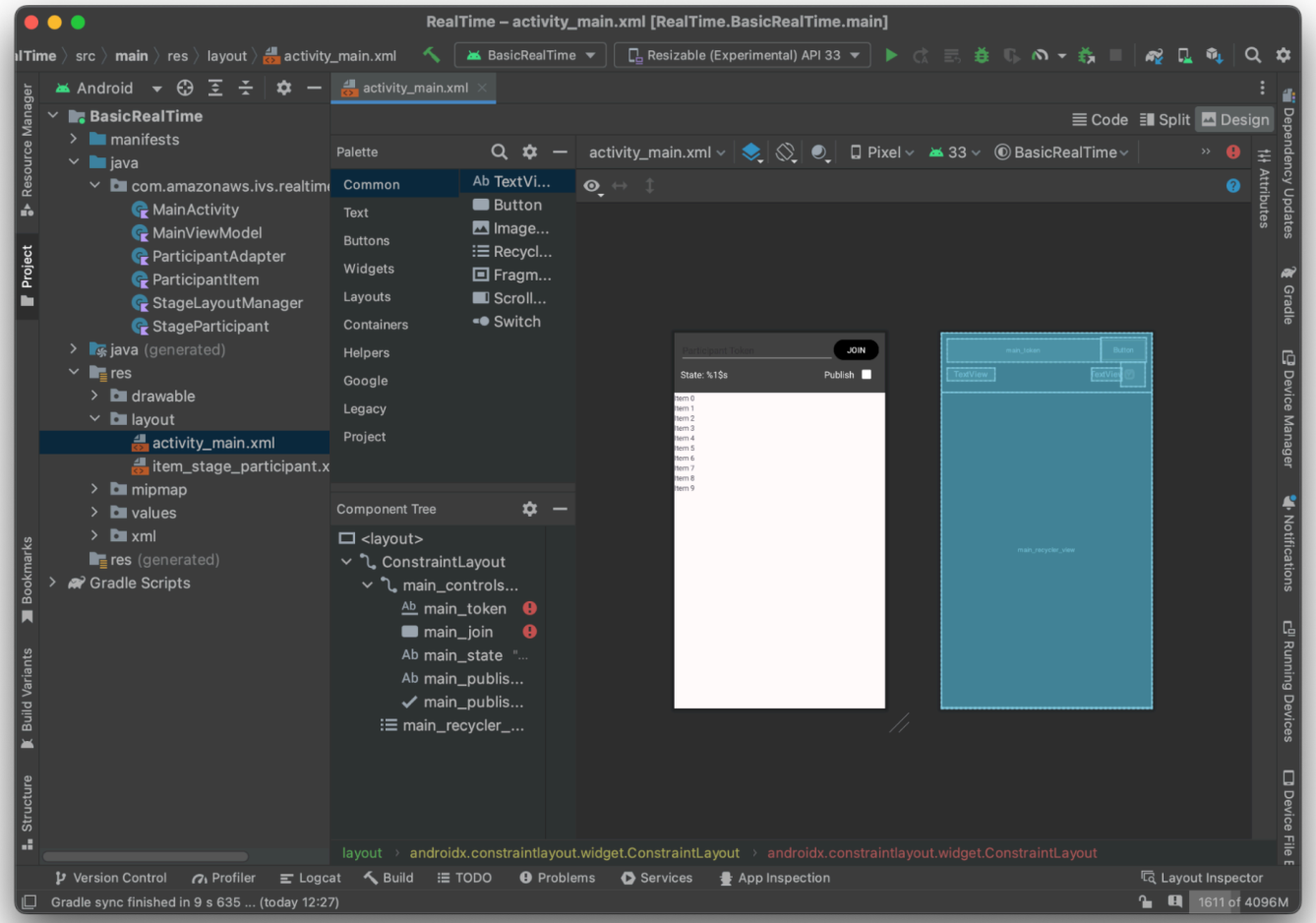
Pasos siguientes

Para ver ejemplos más detallados sobre npm, React y otros, consulta la Guía web del SDK de transmisión de IVS (Guía de transmisión en tiempo real).

Android

Creación de vistas

Empezamos por crear un diseño simple para nuestra aplicación con el archivo `activity_main.xml` creado automáticamente. El diseño contiene `EditText` para agregar un token, un `Button` de unión, `TextView` para mostrar el estado del escenario y `CheckBox` para cambiar la publicación.



Este es el XML que hay detrás de la vista:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:keepScreenOn="true"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".BasicActivity">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:id="@+id/main_controls_container"
            android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:background="@color/cardview_dark_background"
            android:padding="12dp"
            app:layout_constraintTop_toTopOf="parent">

            <EditText
                android:id="@+id/main_token"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:autofillHints="@null"
                android:backgroundTint="@color/white"
                android:hint="@string/token"
                android:imeOptions="actionDone"
                android:inputType="text"
                android:textColor="@color/white"
                app:layout_constraintEnd_toStartOf="@id/main_join"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

            <Button
                android:id="@+id/main_join"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:backgroundTint="@color/black"
                android:text="@string/join"
                android:textAllCaps="true"
                android:textColor="@color/white"
                android:textSize="16sp"
                app:layout_constraintBottom_toBottomOf="@+id/main_token"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toEndOf="@id/main_token" />

            <TextView
                android:id="@+id/main_state"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/state"
                android:textColor="@color/white"
                android:textSize="18sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@id/main_token" />

            <TextView
```

```
                android:id="@+id/main_publish_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/publish"
                android:textColor="@color/white"
                android:textSize="18sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toStartOf="@id/main_publish_checkbox"
                app:layout_constraintTop_toBottomOf="@id/main_token" />

            <CheckBox
                android:id="@+id/main_publish_checkbox"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:buttonTint="@color/white"
                android:checked="true"
                app:layout_constraintBottom_toBottomOf="@id/main_publish_text"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintTop_toTopOf="@id/main_publish_text" />

        </androidx.constraintlayout.widget.ConstraintLayout>

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/main_recycler_view"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            app:layout_constraintTop_toBottomOf="@+id/main_controls_container"
            app:layout_constraintBottom_toBottomOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
<layout>
```

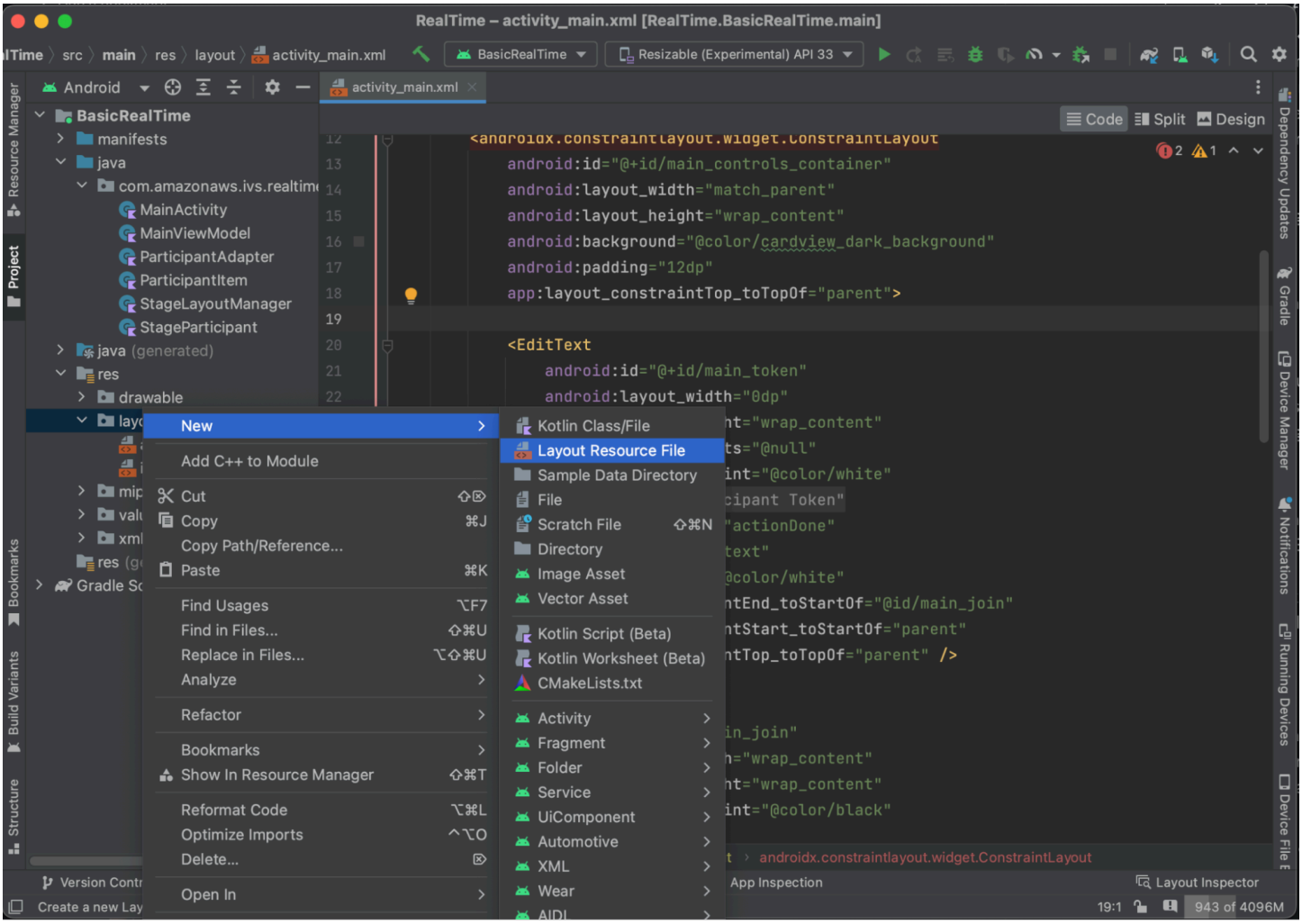
Aquí hacemos referencia a los ID de un par de cadenas, así que es el momento de crear todo el archivo `strings.xml`:

```
<resources>
    <string name="app_name">BasicRealTime</string>
    <string name="join">Join</string>
    <string name="leave">Leave</string>
    <string name="token">Participant Token</string>
    <string name="publish">Publish</string>
    <string name="state">State: %1$s</string>
</resources>
```

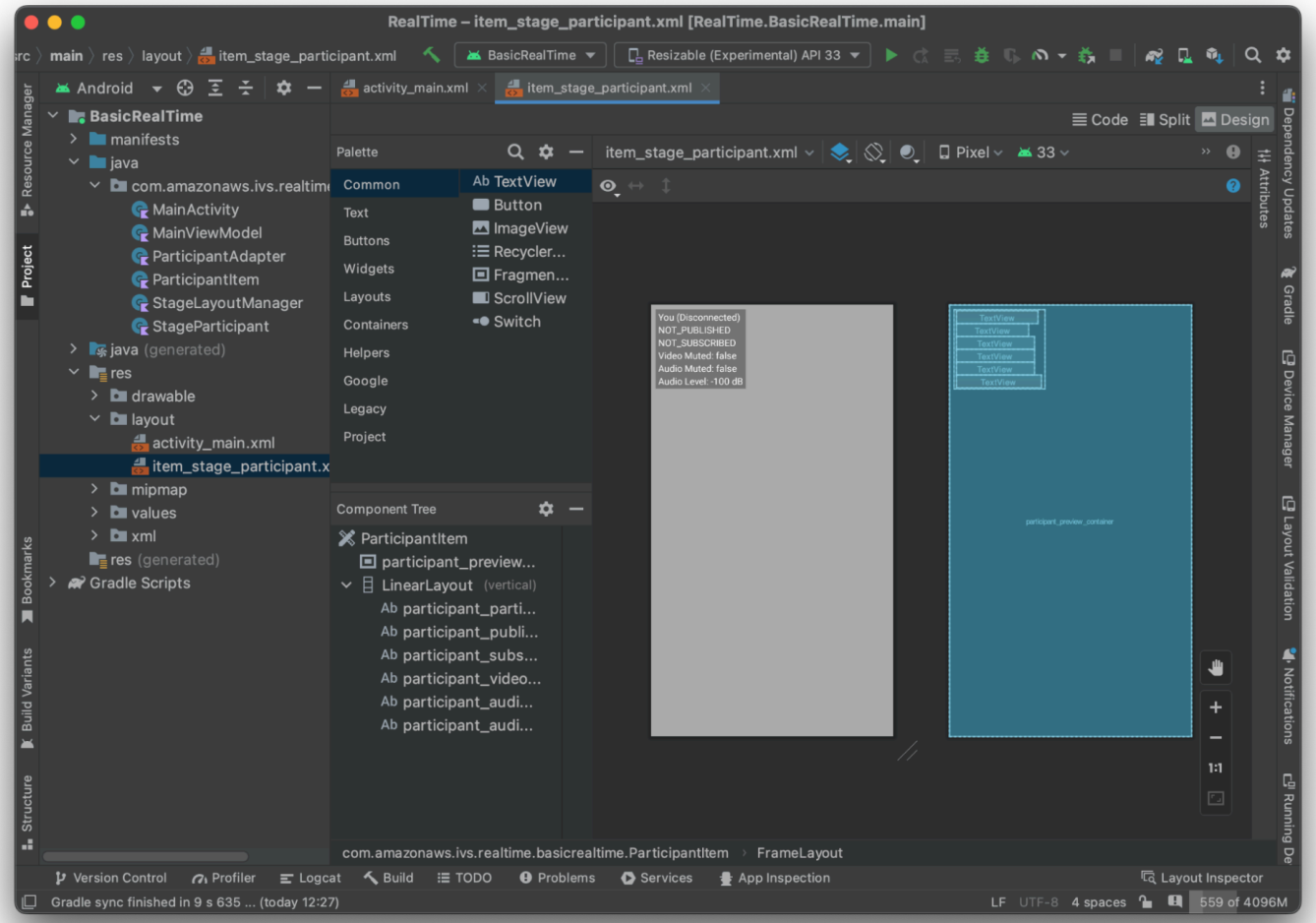
Vamos a vincular esas vistas del XML a nuestro `MainActivity.kt`:

```
import android.widget.Button
import android.widget.CheckBox
import android.widget.EditText
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView

private lateinit var checkboxPublish: CheckBox
private lateinit var recyclerView: RecyclerView
private lateinit var buttonJoin: Button
private lateinit var textViewState: TextView
private lateinit var editTextToken: EditText

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    checkboxPublish = findViewById(R.id.main_publish_checkbox)
    recyclerView = findViewById(R.id.main_recycler_view)
    buttonJoin = findViewById(R.id.main_join)
    textViewState = findViewById(R.id.main_state)
    editTextToken = findViewById(R.id.main_token)
}
```

Ahora crearemos una vista de elementos para `RecyclerView`. Para ello, haga clic con el botón derecho en el directorio `res/layout` y seleccione Nuevo > Archivo de recursos de diseño. Asigne a este archivo el nombre `item_stage_participant.xml`.



El diseño de este elemento es sencillo: contiene una vista para renderizar la transmisión de video de un participante y una lista de etiquetas para mostrar información sobre el participante:



Este es el XML:

```
<?xml version="1.0" encoding="utf-8"?>
<com.amazonaws.ivs.realtime.basicrealtime.ParticipantItem xmlns:android="http://
schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/participant_preview_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:background="@android:color/darker_gray" />
```

```
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="#50000000"
        android:orientation="vertical"
        android:paddingLeft="4dp"
        android:paddingTop="2dp"
        android:paddingRight="4dp"
        android:paddingBottom="2dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/participant_participant_id"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="You (Disconnected)" />

        <TextView
            android:id="@+id/participant_publishing"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="NOT_PUBLISHED" />

        <TextView
            android:id="@+id/participant_subscribed"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="NOT_SUBSCRIBED" />

        <TextView
            android:id="@+id/participant_video_muted"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
```

```
            tools:text="Video Muted: false" />

        <TextView
            android:id="@+id/participant_audio_muted"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="Audio Muted: false" />

        <TextView
            android:id="@+id/participant_audio_level"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            tools:text="Audio Level: -100 dB" />

    </LinearLayout>

</com.amazonaws.ivs.realtime.basicrealtime.ParticipantItem>
```

Este archivo XML infla una clase que aún no hemos creado, `ParticipantItem`. Como el XML incluye el espacio de nombres completo, asegúrese de actualizar este archivo XML en su espacio de nombres. Vamos a crear esta clase y configurar las vistas, aunque lo vamos a dejar en blanco por ahora.

Cree una nueva clase de Kotlin, `ParticipantItem`:

```
package com.amazonaws.ivs.realtime.basicrealtime

import android.content.Context
import android.util.AttributeSet
import android.widget.FrameLayout
import android.widget.TextView
import kotlin.math.roundToInt

class ParticipantItem @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyleAttr: Int = 0,
    defStyleRes: Int = 0,
) : FrameLayout(context, attrs, defStyleAttr, defStyleRes) {
```

```
    private lateinit var previewContainer: FrameLayout
    private lateinit var textViewParticipantId: TextView
    private lateinit var textViewPublish: TextView
    private lateinit var textViewSubscribe: TextView
    private lateinit var textViewVideoMuted: TextView
    private lateinit var textViewAudioMuted: TextView
    private lateinit var textViewAudioLevel: TextView

    override fun onFinishInflate() {
        super.onFinishInflate()
        previewContainer = findViewById(R.id.participant_preview_container)
        textViewParticipantId = findViewById(R.id.participant_participant_id)
        textViewPublish = findViewById(R.id.participant_publishing)
        textViewSubscribe = findViewById(R.id.participant_subscribed)
        textViewVideoMuted = findViewById(R.id.participant_video_muted)
        textViewAudioMuted = findViewById(R.id.participant_audio_muted)
        textViewAudioLevel = findViewById(R.id.participant_audio_level)
    }
}
```

Permisos

Para utilizar la cámara y el micrófono, debe solicitar permisos al usuario. Para ello, seguimos un flujo de permisos estándar:

```
override fun onStart() {
    super.onStart()
    requestPermission()
}

private val requestPermissionLauncher =
    registerForActivityResult(ActivityResultContracts.RequestMultiplePermissions())
 { permissions ->
        if (permissions[Manifest.permission.CAMERA] == true &&
 permissions[Manifest.permission.RECORD_AUDIO] == true) {
            viewModel.permissionGranted() // we will add this later
        }
    }

private val permissions = listOf(
    Manifest.permission.CAMERA,
    Manifest.permission.RECORD_AUDIO,
```

```
)

private fun requestPermission() {
    when {
        this.hasPermissions(permissions) -> viewModel.permissionGranted() // we will
 add this later
        else -> requestPermissionLauncher.launch(permissions.toTypedArray())
    }
}

private fun Context.hasPermissions(permissions: List<String>): Boolean {
    return permissions.all {
        ContextCompat.checkSelfPermission(this, it) ==
 PackageManager.PERMISSION_GRANTED
    }
}
```

Estado de la aplicación

Nuestra aplicación realiza un seguimiento de los participantes a nivel local `MainViewModel.kt` y el estado se comunicará al usuario `MainActivity` mediante Kotlin. StateFlow

Cree una nueva clase de Kotlin (`MainViewModel`):

```
package com.amazonaws.ivs.realtime.basicrealtime

import android.app.Application
import androidx.lifecycle.AndroidViewModel

class MainViewModel(application: Application) : AndroidViewModel(application),
 Stage.Strategy, StageRenderer {

}
```

En `MainActivity.kt`, administramos el modelo de la vista:

```
import androidx.activity.viewModels

private val viewModel: MainViewModel by viewModels()
```

Para usar `AndroidViewModel` y las extensiones `ViewModel` de Kotlin, tendrá que agregar lo siguiente al archivo `build.gradle` del módulo:

```
implementation 'androidx.core:core-ktx:1.10.1'
implementation "androidx.activity:activity-ktx:1.7.2"
implementation 'androidx.appcompat:appcompat:1.6.1'
implementation 'com.google.android.material:material:1.10.0'
implementation "androidx.lifecycle:lifecycle-extensions:2.2.0"

def lifecycle_version = "2.6.1"
implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
```

RecyclerView Adaptador

Crearemos una subclase `RecyclerView.Adapter` sencilla para hacer un seguimiento de nuestros participantes y actualizar `RecyclerView` en los eventos del escenario. Pero antes, necesitamos una clase que represente a un participante. Cree una nueva clase de Kotlin (`StageParticipant`):

```
package com.amazonaws.ivs.realtime.basicrealtime

import com.amazonaws.ivs.broadcast.Stage
import com.amazonaws.ivs.broadcast.StageStream

class StageParticipant(val isLocal: Boolean, var participantId: String?) {
    var publishState = Stage.PublishState.NOT_PUBLISHED
    var subscribeState = Stage.SubscribeState.NOT_SUBSCRIBED
    var streams = mutableListOf<StageStream>()

    val stableID: String
        get() {
            return if (isLocal) {
                "LocalUser"
            } else {
                requireNotNull(participantId)
            }
        }
}
```

Usaremos esta clase en la clase `ParticipantAdapter` que crearemos a continuación. Empezamos por definir la clase y crear una variable para hacer un seguimiento de los participantes:

```
package com.amazonaws.ivs.realtime.basicrealtime
```

```
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView

class ParticipantAdapter : RecyclerView.Adapter<ParticipantAdapter.ViewHolder>() {

    private val participants = mutableListOf<StageParticipant>()
```

También tenemos que definir `RecyclerView.ViewHolder` antes de implementar el resto de las anulaciones:

```
class ViewHolder(val participantItem: ParticipantItem) :
 RecyclerView.ViewHolder(participantItem)
```

Con esto, podemos implementar las anulaciones de `RecyclerView.Adapter` estándar:

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
    val item = LayoutInflater.from(parent.context)
        .inflate(R.layout.item_stage_participant, parent, false) as ParticipantItem
    return ViewHolder(item)
}

override fun getItemCount(): Int {
    return participants.size
}

override fun getItemId(position: Int): Long =
    participants[position]
        .stableID
        .hashCode()
        .toLong()

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    return holder.participantItem.bind(participants[position])
}

override fun onBindViewHolder(holder: ViewHolder, position: Int, payloads:
 MutableList<Any>) {
    val updates = payloads.filterIsInstance<StageParticipant>()
    if (updates.isNotEmpty()) {
        updates.forEach { holder.participantItem.bind(it) // implemented later }
    } else {
        super.onBindViewHolder(holder, position, payloads)
```

```
    }
}
```

Por último, agregamos nuevos métodos a los que llamaremos desde `MainViewModel` cuando se hagan cambios en los participantes. Estos métodos son operaciones CRUD estándar en el adaptador.

```
fun participantJoined(participant: StageParticipant) {
    participants.add(participant)
    notifyItemInserted(participants.size - 1)
}

fun participantLeft(participantId: String) {
    val index = participants.indexOfFirst { it.participantId == participantId }
    if (index != -1) {
        participants.removeAt(index)
        notifyItemRemoved(index)
    }
}

fun participantUpdated(participantId: String?, update: (participant: StageParticipant)
 -> Unit) {
    val index = participants.indexOfFirst { it.participantId == participantId }
    if (index != -1) {
        update(participants[index])
        notifyItemChanged(index, participants[index])
    }
}
```

En `MainViewModel`, tenemos que crear y mantener una referencia a este adaptador:

```
internal val participantAdapter = ParticipantAdapter()
```

Estado de la etapa

También tenemos que hacer un seguimiento del estado de algunos escenarios en `MainViewModel`. Definamos esas propiedades ahora:

```
private val _connectionState = MutableStateFlow(Stage.ConnectionState.DISCONNECTED)
val connectionState = _connectionState.asStateFlow()
```

```
private var publishEnabled: Boolean = false
    set(value) {
        field = value
        // Because the strategy returns the value of `checkboxPublish.isChecked`, just
 call `refreshStrategy`.
        stage?.refreshStrategy()
    }

private var deviceDiscovery: DeviceDiscovery? = null
private var stage: Stage? = null
private var streams = mutableListOf<LocalStageStream>()
```

Para ver su propia vista previa antes de unirse a un escenario, creamos inmediatamente un participante local:

```
init {
    deviceDiscovery = DeviceDiscovery(application)

    // Create a local participant immediately to render our camera preview and
 microphone stats
    val localParticipant = StageParticipant(true, null)
    participantAdapter.participantJoined(localParticipant)
}
```

Queremos asegurarnos de borrar estos recursos cuando se elimine `ViewModel`. Anulamos on`Cleared()` directamente para así no olvidarnos de eliminar estos recursos.

```
override fun onCleared() {
    stage?.release()
    deviceDiscovery?.release()
    deviceDiscovery = null
    super.onCleared()
}
```

A continuación, completamos la propiedad `streams` local nada más recibir los permisos e implementamos el método `permissionsGranted` al que llamamos anteriormente:

```
internal fun permissionGranted() {
    val deviceDiscovery = deviceDiscovery ?: return
    streams.clear()
    val devices = deviceDiscovery.listLocalDevices()
    // Camera
```

```
    devices
        .filter { it.descriptor.type == Device.Descriptor.DeviceType.CAMERA }
        .maxByOrNull { it.descriptor.position == Device.Descriptor.Position.FRONT }
        ?.let { streams.add(ImageLocalStageStream(it)) }
    // Microphone
    devices
        .filter { it.descriptor.type == Device.Descriptor.DeviceType.MICROPHONE }
        .maxByOrNull { it.descriptor.isDefault }
        ?.let { streams.add(AudioLocalStageStream(it)) }

    stage?.refreshStrategy()

    // Update our local participant with these new streams
    participantAdapter.participantUpdated(null) {
        it.streams.clear()
        it.streams.addAll(streams)
    }
}
```

Implementación del SDK de escenarios

Los siguientes tres conceptos básicos subyacen a la funcionalidad de transmisión en tiempo real: escenario, estrategia y renderizador. El objetivo del diseño es minimizar la cantidad de lógica necesaria por parte del cliente para crear un producto que funcione.

Stage.Strategy

La implementación de `Stage.Strategy` es sencilla:

```
override fun stageStreamsToPublishForParticipant(
    stage: Stage,
    participantInfo: ParticipantInfo
): MutableList<LocalStageStream> {
    // Return the camera and microphone to be published.
    // This is only called if `shouldPublishFromParticipant` returns true.
    return streams
}

override fun shouldPublishFromParticipant(stage: Stage, participantInfo:
 ParticipantInfo): Boolean {
    return publishEnabled
}
```

```
override fun shouldSubscribeToParticipant(stage: Stage, participantInfo:
 ParticipantInfo): Stage.SubscribeType {
    // Subscribe to both audio and video for all publishing participants.
    return Stage.SubscribeType.AUDIO_VIDEO
}
```

En resumen, la publicación depende del estado de la variable `publishEnabled` interna y, si publicamos, publicaremos las transmisiones que recopilamos anteriormente. Por último, en este ejemplo, siempre nos suscribimos a otros participantes y recibimos tanto su contenido de audio como de video.

StageRenderer

La implementación de `StageRenderer` también es bastante simple, aunque, dada la cantidad de funciones, contiene bastante más código. El enfoque general de este renderizador es actualizar `ParticipantAdapter` cuando el SDK nos notifique un cambio en un participante. Hay ciertos casos en los que tratamos a los participantes locales de forma diferente, porque hemos decidido administrarlos por nuestra cuenta para que puedan ver la vista previa de la cámara antes de unirse.

```
override fun onError(exception: BroadcastException) {
    Toast.makeText(getApplication(), "onError ${exception.localizedMessage}",
 Toast.LENGTH_LONG).show()
    Log.e("BasicRealTime", "onError $exception")
}

override fun onConnectionStateChanged(
    stage: Stage,
    connectionState: Stage.ConnectionState,
    exception: BroadcastException?
) {
    _connectionState.value = connectionState
}

override fun onParticipantJoined(stage: Stage, participantInfo: ParticipantInfo) {
    if (participantInfo.isLocal) {
        // If this is the local participant joining the stage, update the participant
 with a null ID because we
        // manually added that participant when setting up our preview
        participantAdapter.participantUpdated(null) {
            it.participantId = participantInfo.participantId
        }
    } else {
```

```
        // If they are not local, add them normally
        participantAdapter.participantJoined(
            StageParticipant(
                participantInfo.isLocal,
                participantInfo.participantId
            )
        )
    }
}

override fun onParticipantLeft(stage: Stage, participantInfo: ParticipantInfo) {
    if (participantInfo.isLocal) {
        // If this is the local participant leaving the stage, update the ID but keep
 it around because
        // we want to keep the camera preview active
        participantAdapter.participantUpdated(participantInfo.participantId) {
            it.participantId = null
        }
    } else {
        // If they are not local, have them leave normally
        participantAdapter.participantLeft(participantInfo.participantId)
    }
}

override fun onParticipantPublishStateChanged(
    stage: Stage,
    participantInfo: ParticipantInfo,
    publishState: Stage.PublishState
) {
    // Update the publishing state of this participant
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.publishState = publishState
    }
}

override fun onParticipantSubscribeStateChanged(
    stage: Stage,
    participantInfo: ParticipantInfo,
    subscribeState: Stage.SubscribeState
) {
    // Update the subscribe state of this participant
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.subscribeState = subscribeState
    }
```

```
}

override fun onStreamsAdded(stage: Stage, participantInfo: ParticipantInfo, streams:
 MutableList<StageStream>) {
    // We don't want to take any action for the local participant because we track
 those streams locally
    if (participantInfo.isLocal) {
        return
    }
    // For remote participants, add these new streams to that participant's streams
 array.
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.streams.addAll(streams)
    }
}

override fun onStreamsRemoved(stage: Stage, participantInfo: ParticipantInfo, streams:
 MutableList<StageStream>) {
    // We don't want to take any action for the local participant because we track
 those streams locally
    if (participantInfo.isLocal) {
        return
    }
    // For remote participants, remove these streams from that participant's streams
 array.
    participantAdapter.participantUpdated(participantInfo.participantId) {
        it.streams.removeAll(streams)
    }
}

override fun onStreamsMutedChanged(
    stage: Stage,
    participantInfo: ParticipantInfo,
    streams: MutableList<StageStream>
) {
    // We don't want to take any action for the local participant because we track
 those streams locally
    if (participantInfo.isLocal) {
        return
    }
    // For remote participants, notify the adapter that the participant has been
 updated. There is no need to modify
    // the `streams` property on the `StageParticipant` because it is the same
 `StageStream` instance. Just
```

```
    // query the `isMuted` property again.
    participantAdapter.participantUpdated(participantInfo.participantId) {}
}
```

Implementación de un personalizado RecyclerView LayoutManager

Establecer diferentes números de participantes puede resultar complejo. Lo ideal es que ocupen todo el marco de la vista principal, pero gestionar la configuración de cada participante de forma independiente no es la mejor forma de lograrlo. Para facilitarlo, veremos cómo implementar `RecyclerView.LayoutManager`.

Cree otra clase, `StageLayoutManager`, que debería ampliar `GridLayoutManager`. Esta clase está pensada para calcular cómo se mostrará cada participante en función del número de participantes en un diseño de filas o columnas basado en flujos. Cada fila tiene la misma altura que las demás, pero las columnas pueden tener diferentes anchuras por fila. Consulte el comentario del código que aparece sobre la variable `layouts` para ver cómo personalizar este comportamiento.

```
package com.amazonaws.ivs.realtime.basicrealtime

import android.content.Context
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.RecyclerView

class StageLayoutManager(context: Context?) : GridLayoutManager(context, 6) {

    companion object {
        /**
         * This 2D array contains the description of how the grid of participants
 should be rendered
         * The index of the 1st dimension is the number of participants needed to
 active that configuration
         * Meaning if there is 1 participant, index 0 will be used. If there are 5
 participants, index 4 will be used.
         *
         * The 2nd dimension is a description of the layout. The length of the array is
 the number of rows that
         * will exist, and then each number within that array is the number of columns
 in each row.
         *
         * See the code comments next to each index for concrete examples.
         *
         * This can be customized to fit any layout configuration needed.
```

```
         */
        val layouts: List<List<Int>> = listOf(
            // 1 participant
            listOf(1), // 1 row, full width
            // 2 participants
            listOf(1, 1), // 2 rows, all columns are full width
            // 3 participants
            listOf(1, 2), // 2 rows, first row's column is full width then 2nd row's
columns are 1/2 width
            // 4 participants
            listOf(2, 2), // 2 rows, all columns are 1/2 width
            // 5 participants
            listOf(1, 2, 2), // 3 rows, first row's column is full width, 2nd and 3rd
row's columns are 1/2 width
            // 6 participants
            listOf(2, 2, 2), // 3 rows, all column are 1/2 width
            // 7 participants
            listOf(2, 2, 3), // 3 rows, 1st and 2nd row's columns are 1/2 width, 3rd
row's columns are 1/3rd width
            // 8 participants
            listOf(2, 3, 3),
            // 9 participants
            listOf(3, 3, 3),
            // 10 participants
            listOf(2, 3, 2, 3),
            // 11 participants
            listOf(2, 3, 3, 3),
            // 12 participants
            listOf(3, 3, 3, 3),
        )
    }

    init {
        spanSizeLookup = object : SpanSizeLookup() {
            override fun getSpanSize(position: Int): Int {
                if (itemCount <= 0) {
                    return 1
                }
                // Calculate the row we're in
                val config = layouts[itemCount - 1]
                var row = 0
                var curPosition = position
                while (curPosition - config[row] >= 0) {
                    curPosition -= config[row]
```

```
                    row++
                }
                // spanCount == max spans, config[row] = number of columns we want
                // So spanCount / config[row] would be something like 6 / 3 if we want
 3 columns.
                // So this will take up 2 spans, with a max of 6 is 1/3rd of the view.
                return spanCount / config[row]
            }
        }
    }

    override fun onLayoutChildren(recycler: RecyclerView.Recycler?, state:
 RecyclerView.State?) {
        if (itemCount <= 0 || state?.isPreLayout == true) return

        val parentHeight = height
        val itemHeight = parentHeight / layouts[itemCount - 1].size // height divided
 by number of rows.

        // Set the height of each view based on how many rows exist for the current
 participant count.
        for (i in 0 until childCount) {
            val child = getChildAt(i) ?: continue
            val layoutParams = child.layoutParams as RecyclerView.LayoutParams
            if (layoutParams.height != itemHeight) {
                layoutParams.height = itemHeight
                child.layoutParams = layoutParams
            }
        }
        // After we set the height for all our views, call super.
        // This works because our RecyclerView can not scroll and all views are always
 visible with stable IDs.
        super.onLayoutChildren(recycler, state)
    }

    override fun canScrollVertically(): Boolean = false
    override fun canScrollHorizontally(): Boolean = false
}
```

En `MainActivity.kt`, tenemos que configurar el adaptador y el administrador de diseño de `RecyclerView`:

```
// In onCreate after setting recyclerView.
```

```
recyclerView.layoutManager = StageLayoutManager(this)
recyclerView.adapter = viewModel.participantAdapter
```

Enlace de acciones de la interfaz de usuario

Ya casi está, solo tenemos que enlazar algunas acciones de la interfaz de usuario.

Primero, tenemos que hacer que `MainActivity` supervise los cambios en `StateFlow` desde `MainViewModel`:

```
// At the end of your onCreate method
lifecycleScope.launch {
    repeatOnLifecycle(Lifecycle.State.CREATED) {
        viewModel.connectionState.collect { state ->
            buttonJoin.setText(if (state == ConnectionState.DISCONNECTED) R.string.join
 else R.string.leave)
            textViewState.text = getString(R.string.state, state.name)
        }
    }
}
```

A continuación, agregamos los oyentes a nuestro botón de unión y a la casilla de publicación:

```
buttonJoin.setOnClickListener {
    viewModel.joinStage(editTextToken.text.toString())
}
checkboxPublish.setOnCheckedChangeListener { _, isChecked ->
    viewModel.setPublishEnabled(isChecked)
}
```

Ambos elementos llaman a funciones en `MainViewModel`, lo cual implementaremos ahora:

```
internal fun joinStage(token: String) {
    if (_connectionState.value != Stage.ConnectionState.DISCONNECTED) {
        // If we're already connected to a stage, leave it.
        stage?.leave()
    } else {
        if (token.isEmpty()) {
            Toast.makeText(getApplication(), "Empty Token", Toast.LENGTH_SHORT).show()
            return
        }
        try {
```

```
            // Destroy the old stage first before creating a new one.
            stage?.release()
            val stage = Stage(getApplication(), token, this)
            stage.addRenderer(this)
            stage.join()
            this.stage = stage
        } catch (e: BroadcastException) {
            Toast.makeText(getApplication(), "Failed to join stage
 ${e.localizedMessage}", Toast.LENGTH_LONG).show()
            e.printStackTrace()
        }
    }
}

internal fun setPublishEnabled(enabled: Boolean) {
    publishEnabled = enabled
}
```

Renderización de los participantes

Por último, tenemos que renderizar los datos que recibimos del SDK en el elemento de participante que creamos anteriormente. Ya hemos completado la lógica de `RecyclerView`, por lo que solo tenemos que implementar la API `bind` en `ParticipantItem`.

Empezamos por agregar la función vacía y, luego, la analizaremos paso a paso:

```
fun bind(participant: StageParticipant) {

}
```

Primero, analizaremos el estado sencillo, el ID del participante, el estado de la publicación y el estado de la suscripción. Para hacerlo, tan solo actualizamos `TextViews` directamente:

```
val participantId = if (participant.isLocal) {
    "You (${participant.participantId ?: "Disconnected"})"
} else {
    participant.participantId
}
textViewParticipantId.text = participantId
textViewPublish.text = participant.publishState.name
textViewSubscribe.text = participant.subscribeState.name
```

A continuación, actualizaremos los estados silenciados de audio y video. Para obtener el estado silenciado, tenemos que encontrar el `ImageDevice` y el `AudioDevice` de la matriz de transmisiones. Para optimizar el rendimiento, recordamos los últimos ID de los dispositivos conectados.

```
// This belongs outside the `bind` API.
private var imageDeviceUrn: String? = null
private var audioDeviceUrn: String? = null

// This belongs inside the `bind` API.
val newImageStream = participant
    .streams
    .firstOrNull { it.device is ImageDevice }
textViewVideoMuted.text = if (newImageStream != null) {
    if (newImageStream.muted) "Video muted" else "Video not muted"
} else {
    "No video stream"
}

val newAudioStream = participant
    .streams
    .firstOrNull { it.device is AudioDevice }
textViewAudioMuted.text = if (newAudioStream != null) {
    if (newAudioStream.muted) "Audio muted" else "Audio not muted"
} else {
    "No audio stream"
}
```

Por último, queremos renderizar una vista previa de `imageDevice`:

```
if (newImageStream?.device?.descriptor?.urn != imageDeviceUrn) {
    // If the device has changed, remove all subviews from the preview container
    previewContainer.removeAllViews()
    (newImageStream?.device as? ImageDevice)?.let {
        val preview = it.getPreviewView(BroadcastConfiguration.AspectMode.FIT)
        previewContainer.addView(preview)
        preview.layoutParams = FrameLayout.LayoutParams(
            FrameLayout.LayoutParams.MATCH_PARENT,
            FrameLayout.LayoutParams.MATCH_PARENT
        )
    }
}
```

```
imageDeviceUrn = newImageStream?.device?.descriptor?.urn
```

Y mostramos las estadísticas de audio de `audioDevice`:

```
if (newAudioStream?.device?.descriptor?.urn != audioDeviceUrn) {
    (newAudioStream?.device as? AudioDevice)?.let {
        it.setStatsCallback { _, rms ->
            textViewAudioLevel.text = "Audio Level: ${rms.roundToInt()} dB"
        }
    }
}
audioDeviceUrn = newAudioStream?.device?.descriptor?.urn
```

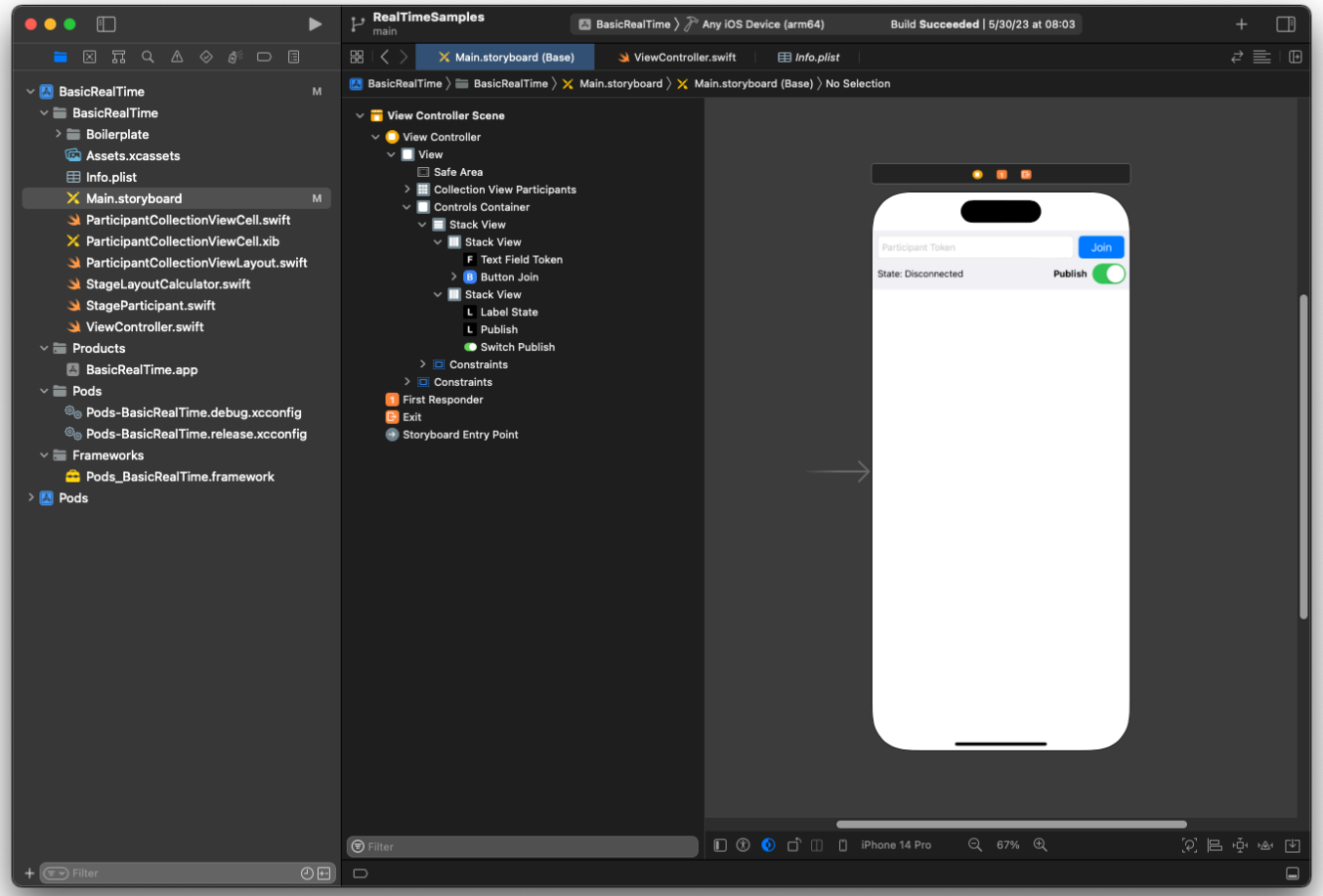
iOS

Creación de vistas

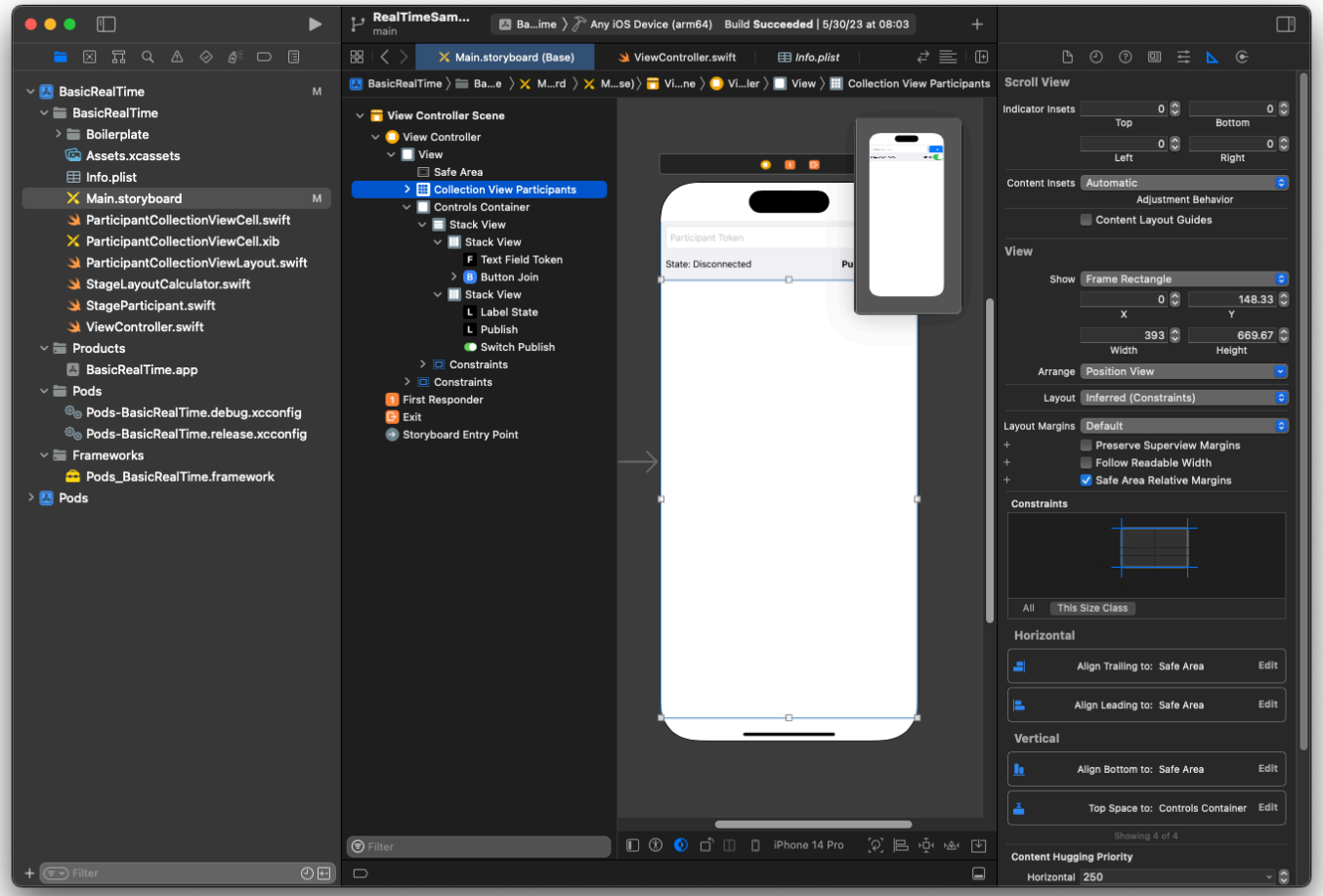
Lo primero es usar el archivo `ViewController.swift` creado automáticamente para importar `AmazonIVSBroadcast` y, luego, agregar algunas `@IBOutlets` que vincular:

```
import AmazonIVSBroadcast

class ViewController: UIViewController {

    @IBOutlet private var textFieldToken: UITextField!
    @IBOutlet private var buttonJoin: UIButton!
    @IBOutlet private var labelState: UILabel!
    @IBOutlet private var switchPublish: UISwitch!
    @IBOutlet private var collectionViewParticipants: UICollectionView!
```

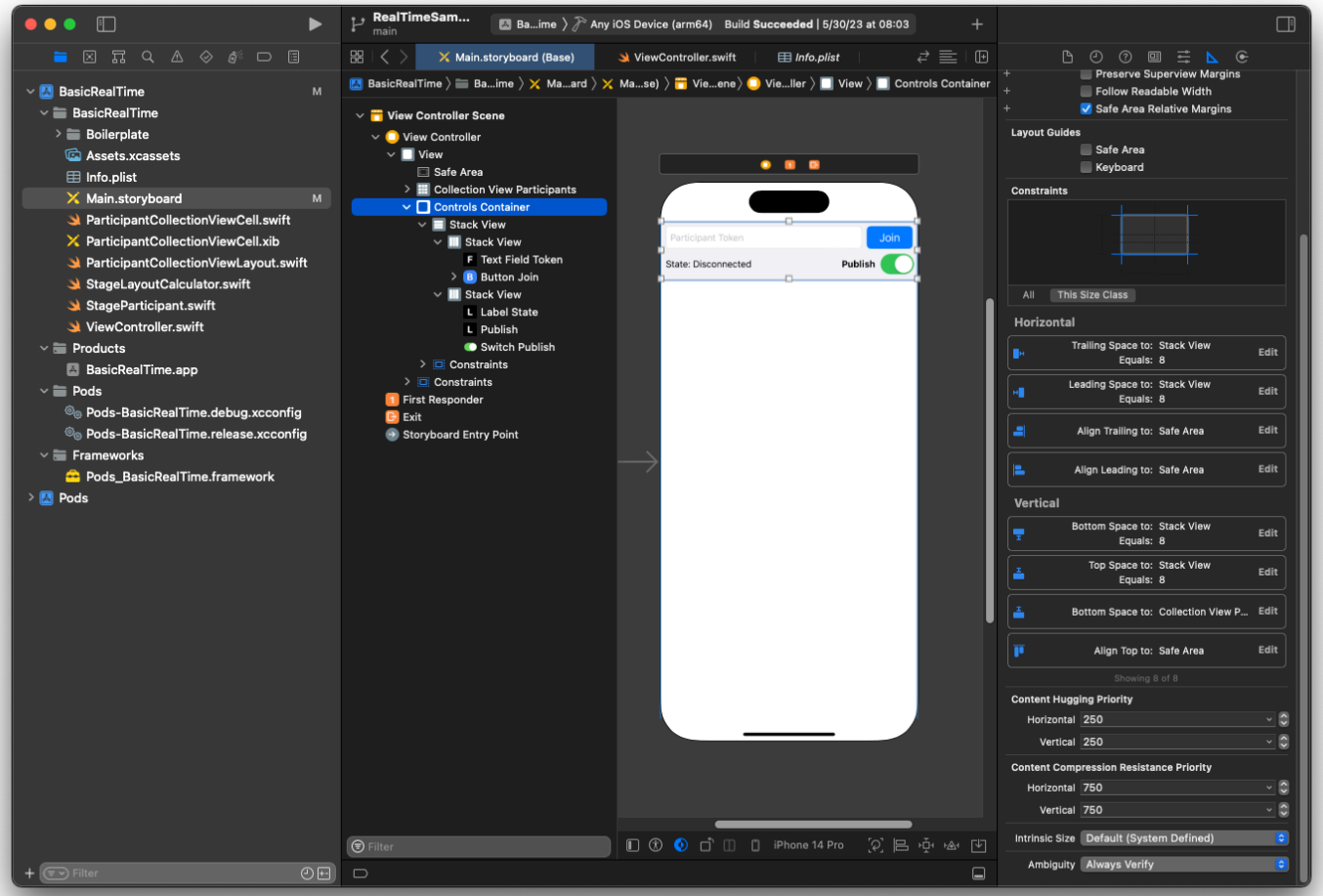
Ahora creamos esas vistas y las vinculamos en `Main.storyboard`. Esta es la estructura de vistas que utilizaremos:



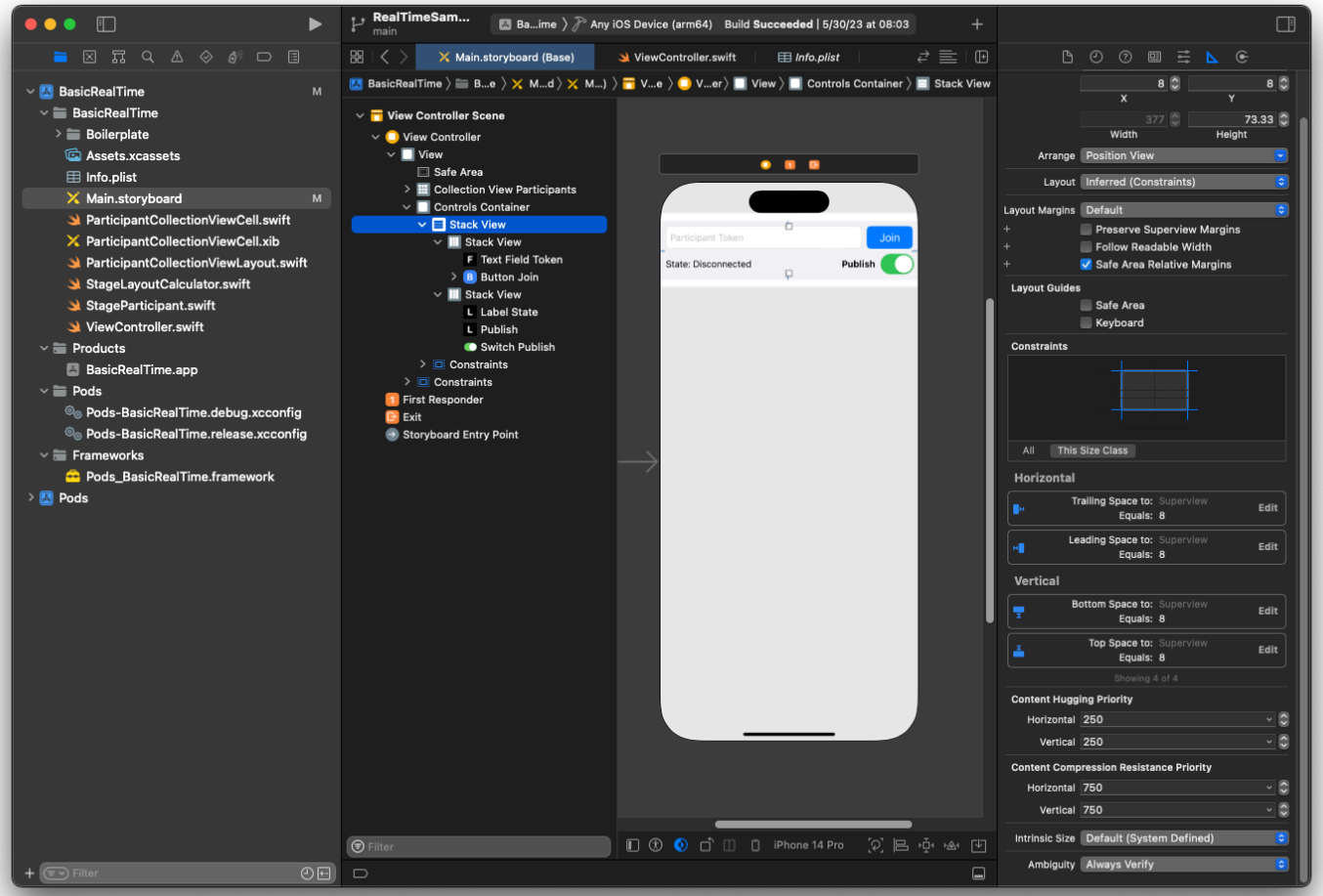
Para AutoLayout la configuración, necesitamos personalizar tres vistas. La primera vista es Vista de colección de participantes (una `UICollectionView`). Alinee Inicial, Final e Inferior con respecto a Área segura. Alinee también Superior con respecto a Contenedor de controles.



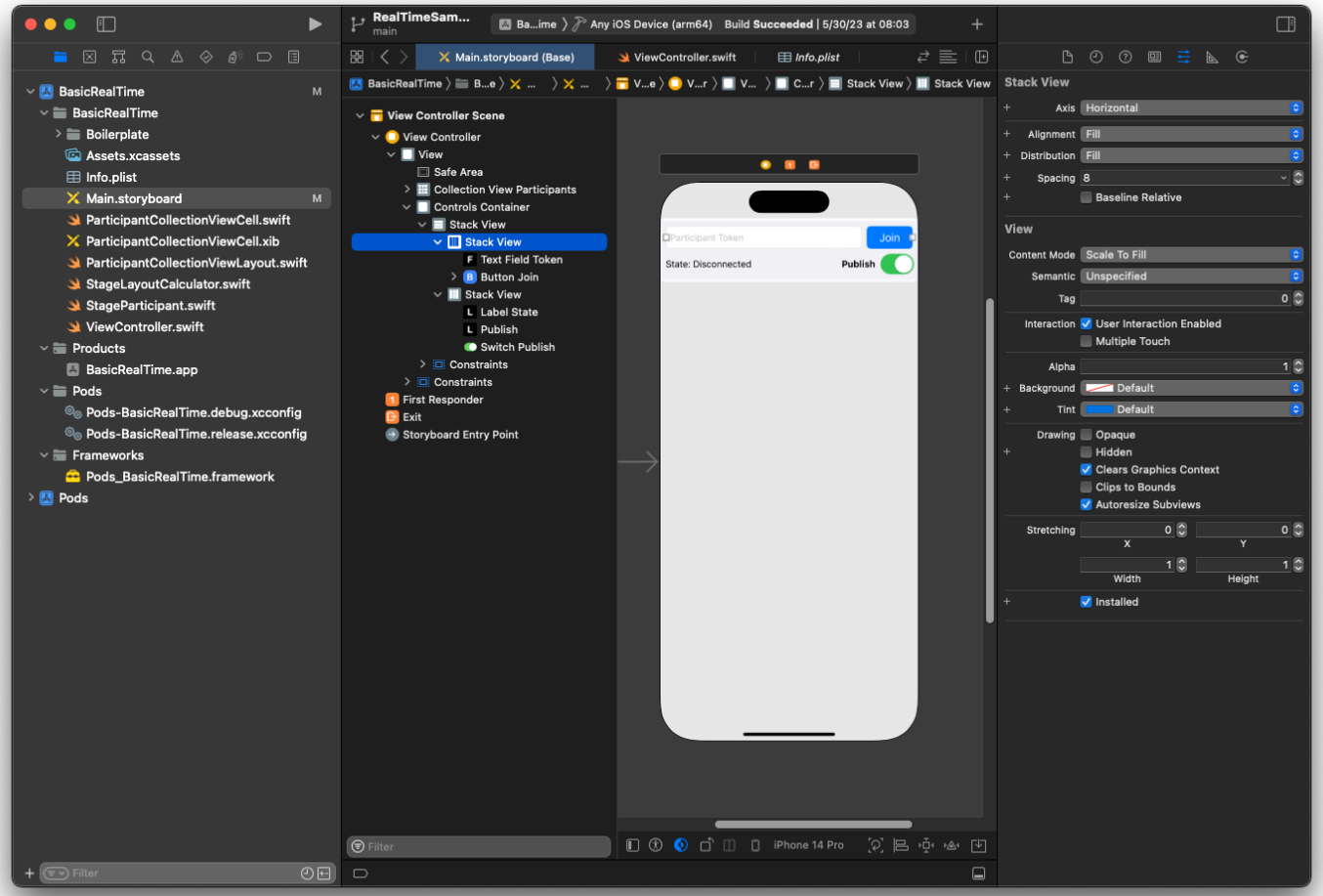
La segunda vista es Contenedor de controles. Alinee Inicial, Final y Superior con respecto a Área segura:



La tercera y última vista es Vista de pila vertical. Alinee Superior, Inicial, Final e Inferior con respecto a Supervista. En cuanto al diseño, establezca el espaciado en 8 en lugar de en 0.



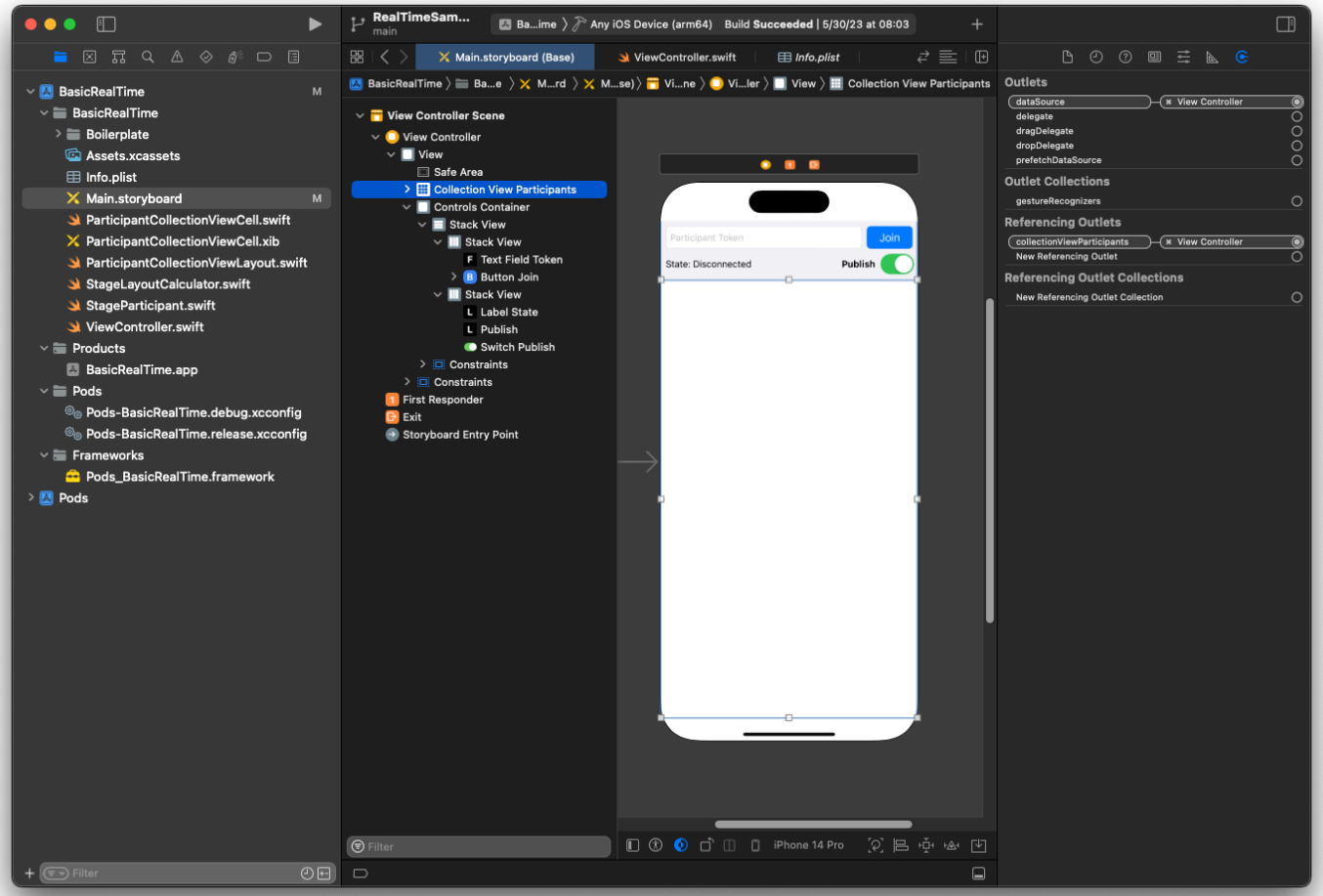
La interfaz de usuario StackViews se encargará del diseño de las vistas restantes. Para las tres interfaces de usuario StackViews, utilice Rellenar como alineación y distribución.



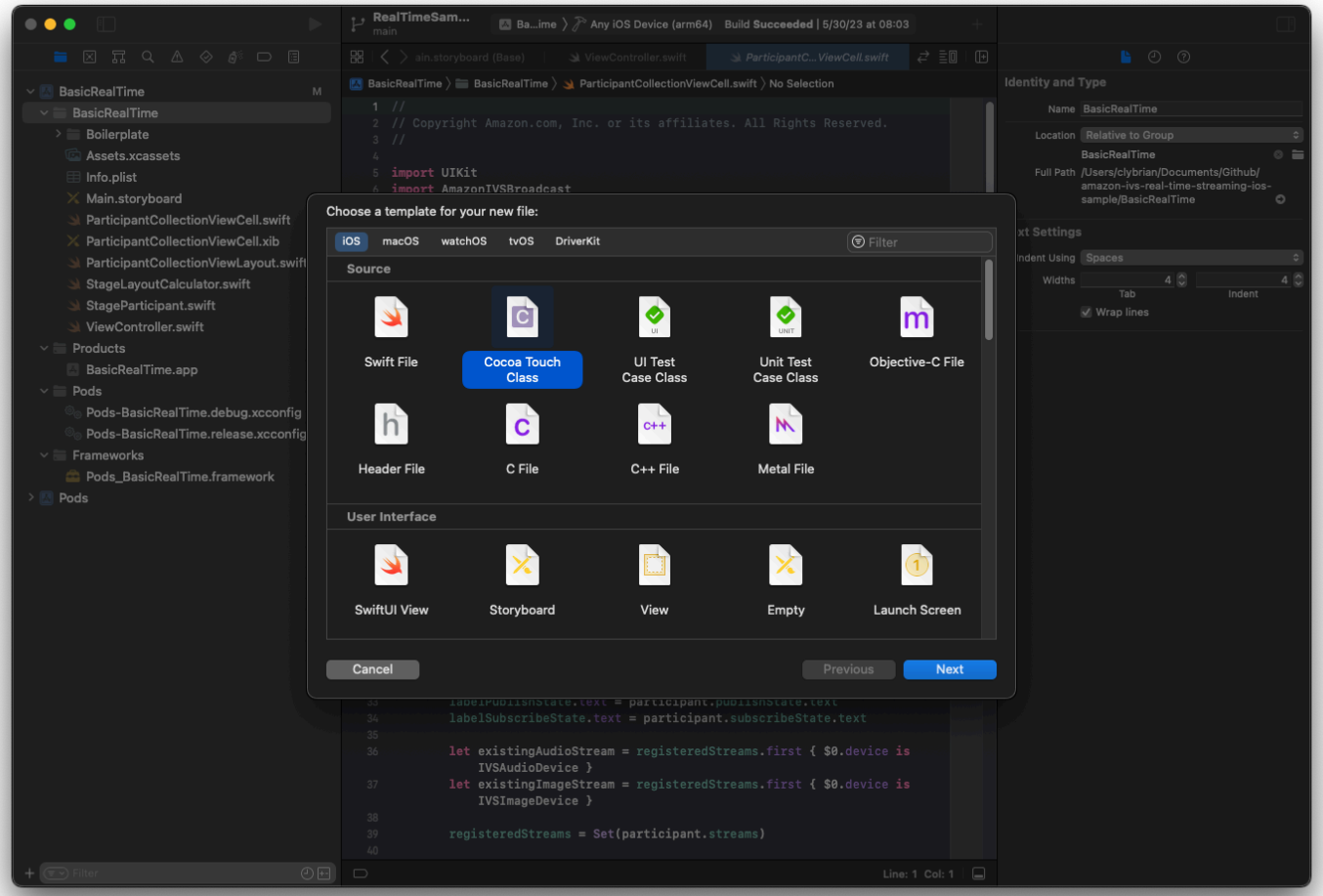
Por último, vinculemos estas vistas a nuestro `ViewController`. Desde arriba, asigne las siguientes vistas:

- El campo de texto de unión se asigna a `textFieldToken`.
- El botón de unión se asigna a `buttonJoin`.
- La etiqueta de estado se asigna a `labelState`.
- El botón de publicación se asigna a `switchPublish`.
- Vista de colección de participantes se asigna a `collectionViewParticipants`.

Aproveche también para establecer el `dataSource` del elemento Vista de colección de participantes en el `ViewController` propietario:



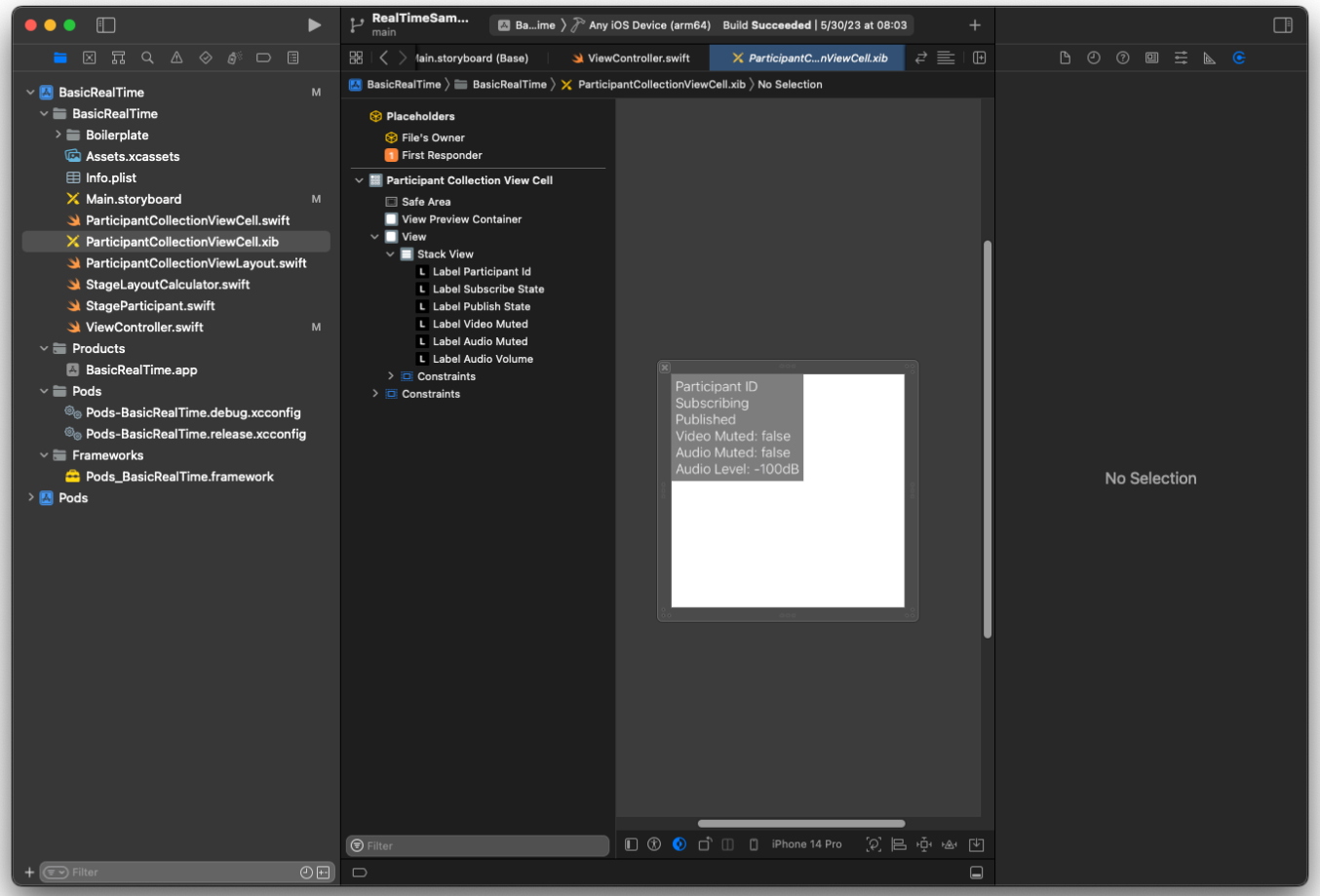
Ahora creamos la subclase `UICollectionViewCell` en la que se renderizarán los participantes. Lo primero es crear un nuevo archivo Cocoa Touch Class:



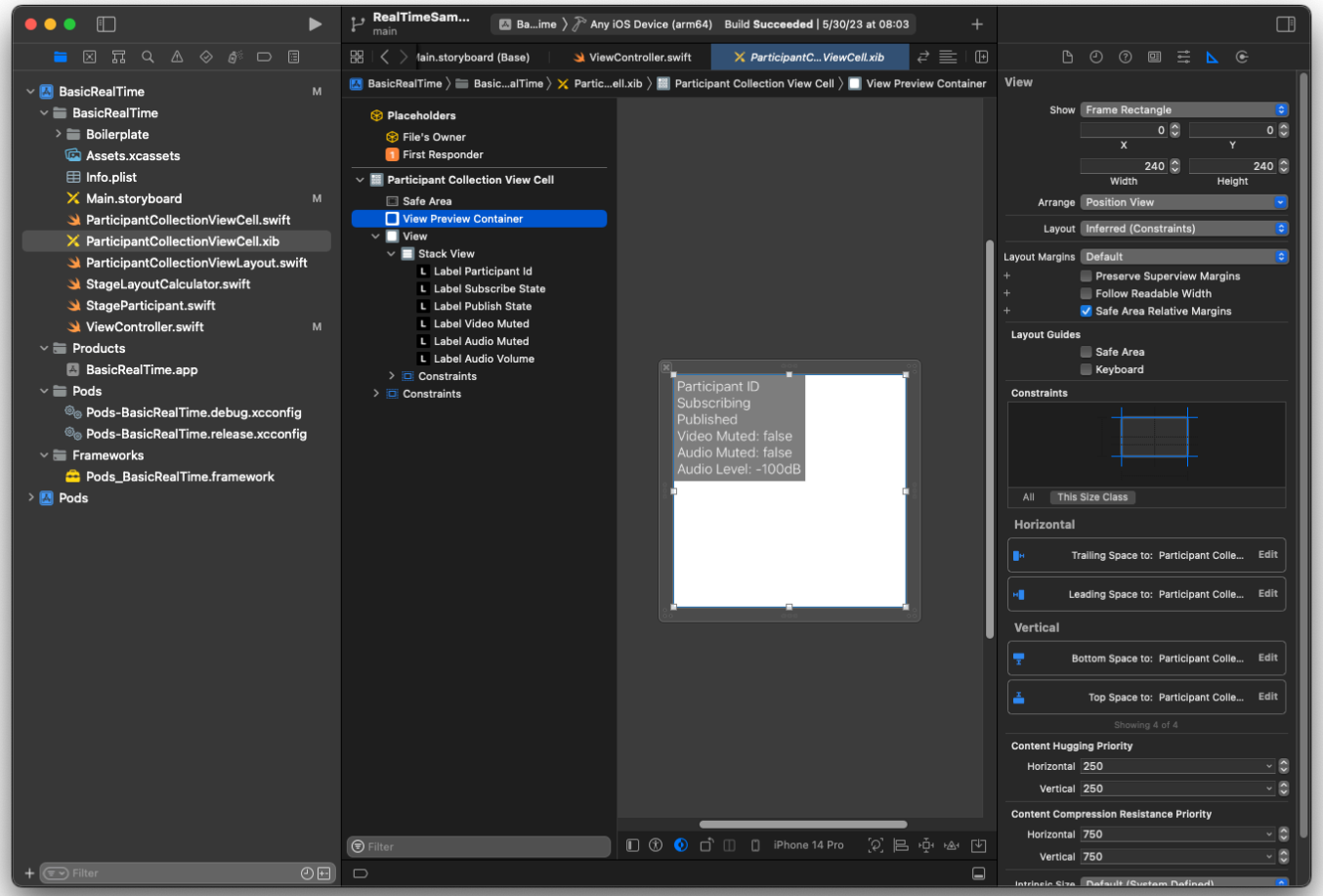
Asígnele el nombre `ParticipantUICollectionViewCell` y conviértala en una subclase de `UICollectionViewCell` en Swift. Volvemos al archivo de Swift y creamos `@IBOutlets` que vincular:

```
import AmazonIVSBroadcast

class ParticipantCollectionViewCell: UICollectionViewCell {

    @IBOutlet private var viewPreviewContainer: UIView!
    @IBOutlet private var labelParticipantId: UILabel!
    @IBOutlet private var labelSubscribeState: UILabel!
    @IBOutlet private var labelPublishState: UILabel!
    @IBOutlet private var labelVideoMuted: UILabel!
    @IBOutlet private var labelAudioMuted: UILabel!
    @IBOutlet private var labelAudioVolume: UILabel!
```

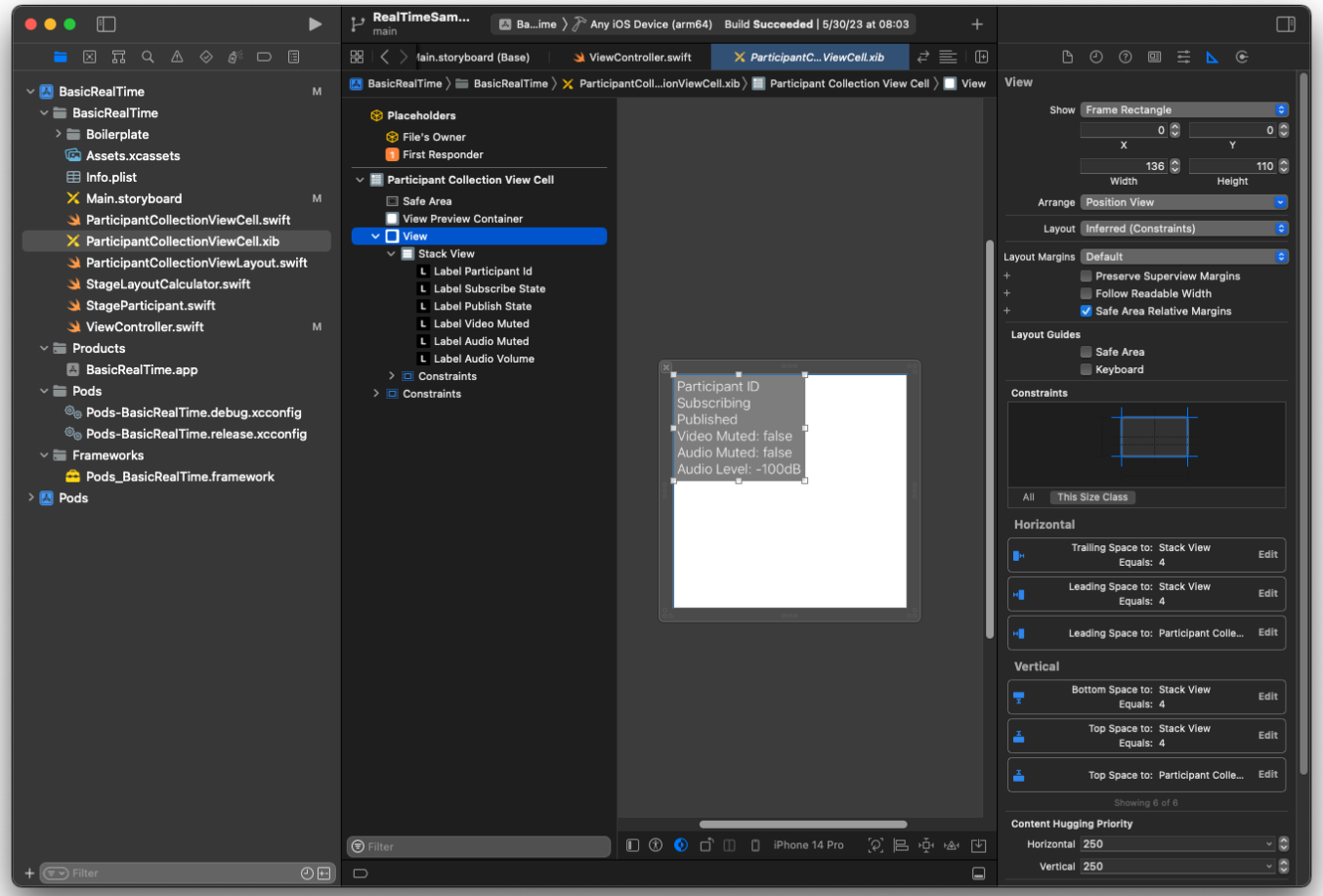
En el archivo XIB asociado, cree esta jerarquía de vistas:



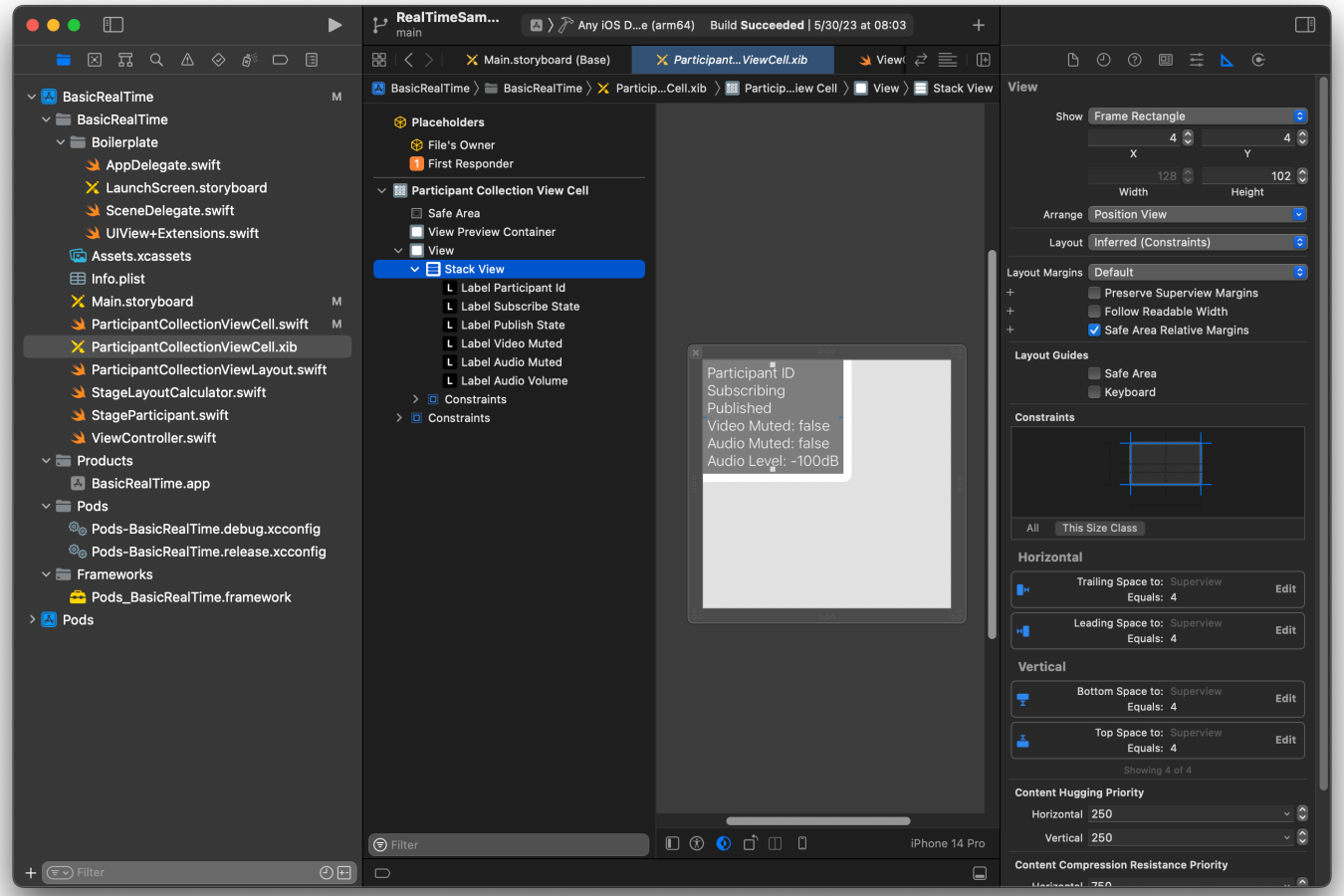
Para AutoLayout ello, volveremos a modificar tres vistas. La primera vista es Vista previa de contenedores. Alinee Final, Inicial, Superior e Inferior con respecto a Celda de vista de colección de participantes.



La segunda vista es Vista. Alinee Inicial y Superior con respecto a Celda de vista de colección de participantes y cambie el valor a 4.



La tercera vista es Vista de pila. Alinee Final, Inicial, Superior e Inferior con respecto a Supervista y cambie el valor a 4.



Permisos y temporizador de inactividad

En `ViewController`, tenemos que desactivar el temporizador de inactividad del sistema para evitar que el dispositivo entre en estado de suspensión mientras se esté utilizando nuestra aplicación:

```
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    // Prevent the screen from turning off during a call.
    UIApplication.shared.isIdleTimerDisabled = true
}

override func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    UIApplication.shared.isIdleTimerDisabled = false
}
```

A continuación, solicitamos los permisos de cámara y micrófono del sistema:

```
private func checkPermissions() {
    checkOrGetPermission(for: .video) { [weak self] granted in
        guard granted else {
            print("Video permission denied")
            return
        }
        self?.checkOrGetPermission(for: .audio) { [weak self] granted in
            guard granted else {
                print("Audio permission denied")
                return
            }
            self?.setupLocalUser() // we will cover this later
        }
    }
}

private func checkOrGetPermission(for mediaType: AVMediaType, _ result: @escaping
 (Bool) -> Void) {
    func mainThreadResult(_ success: Bool) {
        DispatchQueue.main.async {
            result(success)
        }
    }
    switch AVCaptureDevice.authorizationStatus(for: mediaType) {
    case .authorized: mainThreadResult(true)
    case .notDetermined:
        AVCaptureDevice.requestAccess(for: mediaType) { granted in
            mainThreadResult(granted)
        }
    case .denied, .restricted: mainThreadResult(false)
    @unknown default: mainThreadResult(false)
    }
}
```

Estado de la aplicación

Tenemos que configurar `collectionViewParticipants` con el archivo de diseño que creamos anteriormente:

```
override func viewDidLoad() {
    super.viewDidLoad()
    // We render everything to exactly the frame, so don't allow scrolling.
    collectionViewParticipants.isScrollEnabled = false
```

```
    collectionViewParticipants.register(UINib(nibName: "ParticipantCollectionViewCell",
 bundle: .main), forCellWithReuseIdentifier: "ParticipantCollectionViewCell")
}
```

Para representar a cada participante, creamos una estructura simple llamada `StageParticipant`. Esta puede incluirse en el archivo `ViewController.swift`, aunque también se puede crear un archivo nuevo.

```
import Foundation
import AmazonIVSBroadcast

struct StageParticipant {
    let isLocal: Bool
    var participantId: String?
    var publishState: IVSParticipantPublishState = .notPublished
    var subscribeState: IVSParticipantSubscribeState = .notSubscribed
    var streams: [IVSStageStream] = []

    init(isLocal: Bool, participantId: String?) {
        self.isLocal = isLocal
        self.participantId = participantId
    }
}
```

Para hacer un seguimiento de esos participantes, los guardamos en una matriz que sea una propiedad privada de `ViewController`:

```
private var participants = [StageParticipant]()
```

Esta propiedad se utilizará para facilitar información al `UICollectionViewDataSource` que enlazamos en el guion gráfico anterior:

```
extension ViewController: UICollectionViewDataSource {

    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection
 section: Int) -> Int {
        return participants.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
 IndexPath) -> UICollectionViewCell {
```

```
        if let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
 "ParticipantCollectionViewCell", for: indexPath) as? ParticipantCollectionViewCell {
            cell.set(participant: participants[indexPath.row])
            return cell
        } else {
            fatalError("Couldn't load custom cell type
 'ParticipantCollectionViewCell'")
        }
    }

}
```

Para ver su propia vista previa antes de unirse a un escenario, creamos inmediatamente un participante local:

```
override func viewDidLoad() {
    /* existing UICollectionView code */
    participants.append(StageParticipant(isLocal: true, participantId: nil))
}
```

Esto hace que la celda de un participante se renderice inmediatamente cuando se ejecute la aplicación. La celda representa al participante local.

Los usuarios quieren poder verse antes de unirse a un escenario, por lo que, a continuación, implementamos el método `setupLocalUser()`. El código de gestión de permisos anterior se encarga de llamar a este método. Almacenamos la referencia de la cámara y el micrófono como objetos de `IVSLocalStageStream`.

```
private var streams = [IVSLocalStageStream]()
private let deviceDiscovery = IVSDeviceDiscovery()

private func setupLocalUser() {
    // Gather our camera and microphone once permissions have been granted
    let devices = deviceDiscovery.listLocalDevices()
    streams.removeAll()
    if let camera = devices.compactMap({ $0 as? IVSCamera }).first {
        streams.append(IVSLocalStageStream(device: camera))
        // Use a front camera if available.
        if let frontSource = camera.listAvailableInputSources().first(where:
 { $0.position == .front }) {
            camera.setPreferredInputSource(frontSource)
        }
```

```
    }
    if let mic = devices.compactMap({ $0 as? IVSMicrophone }).first {
        streams.append(IVSLocalStageStream(device: mic))
    }
    participants[0].streams = streams
    participantsChanged(index: 0, changeType: .updated)
}
```

Con esto detectamos la cámara y el micrófono del dispositivo a través del SDK y los almacenamos en nuestro objeto `streams` local. A continuación, asignamos la matriz `streams` del primer participante (el participante local que creamos anteriormente) a `streams`. Por último, llamamos a `participantsChanged` con un `index` de 0 y un `changeType` con el valor `updated`. Esta es una función auxiliar para actualizar `UICollectionView` con bonitas animaciones. Este es el resultado:

```
private func participantsChanged(index: Int, changeType: ChangeType) {
    switch changeType {
    case .joined:
        collectionViewParticipants?.insertItems(at: [IndexPath(item: index, section:
 0)])
    case .updated:
        // Instead of doing reloadItems, just grab the cell and update it ourselves. It
 saves a create/destroy of a cell
        // and more importantly fixes some UI flicker. We disable scrolling so the
 index path per cell
        // never changes.
        if let cell = collectionViewParticipants?.cellForItem(at: IndexPath(item:
 index, section: 0)) as? ParticipantCollectionViewCell {
            cell.set(participant: participants[index])
        }
    case .left:
        collectionViewParticipants?.deleteItems(at: [IndexPath(item: index, section:
 0)])
    }
}
```

No se preocupe por `cell.set` aún, lo veremos luego, pero ahí es donde renderizaremos el contenido de la celda en función del participante.

`ChangeType` es una enumeración simple:

```
enum ChangeType {
```

```
    case joined, updated, left
}
```

Lo último es ver si el escenario está conectado. Usamos un simple valor `bool` para comprobarlo. Cuando este valor se actualice, actualizará automáticamente nuestra interfaz de usuario.

```
private var connectingOrConnected = false {
    didSet {
        buttonJoin.setTitle(connectingOrConnected ? "Leave" : "Join", for: .normal)
        buttonJoin.tintColor = connectingOrConnected ? .systemRed : .systemBlue
    }
}
```

Implementación del SDK de escenarios

Los siguientes tres conceptos básicos subyacen a la funcionalidad de transmisión en tiempo real: escenario, estrategia y renderizador. El objetivo del diseño es minimizar la cantidad de lógica necesaria por parte del cliente para crear un producto que funcione.

IVS StageStrategy

La implementación de `IVSStageStrategy` es sencilla:

```
extension ViewController: IVSStageStrategy {
    func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream] {
        // Return the camera and microphone to be published.
        // This is only called if `shouldPublishParticipant` returns true.
        return streams
    }

    func stage(_ stage: IVSStage, shouldPublishParticipant participant:
 IVSParticipantInfo) -> Bool {
        // Our publish status is based directly on the UISwitch view
        return switchPublish.isOn
    }

    func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
        // Subscribe to both audio and video for all publishing participants.
        return .audioVideo
```

```
    }
}
```

En resumen, solo publicamos si la opción de publicación está activada. Si publicamos, publicaremos las transmisiones que hemos recopilado anteriormente. Por último, en este ejemplo, siempre nos suscribimos a otros participantes y recibimos tanto su contenido de audio como de video.

IVS StageRenderer

La implementación de `IVSStageRenderer` también es bastante simple, aunque, dada la cantidad de funciones, contiene bastante más código. El enfoque general de este renderizador es actualizar la matriz `participants` cuando el SDK nos notifique un cambio en un participante. Hay ciertos casos en los que tratamos a los participantes locales de forma diferente, porque hemos decidido administrarlos por nuestra cuenta para que puedan ver la vista previa de la cámara antes de unirse.

```
extension ViewController: IVSStageRenderer {

    func stage(_ stage: IVSStage, didChange connectionState: IVSStageConnectionState,
 withError error: Error?) {
        labelState.text = connectionState.text
        connectingOrConnected = connectionState != .disconnected
    }

    func stage(_ stage: IVSStage, participantDidJoin participant: IVSParticipantInfo) {
        if participant.isLocal {
            // If this is the local participant joining the Stage, update the first
 participant in our array because we
            // manually added that participant when setting up our preview
            participants[0].participantId = participant.participantId
            participantsChanged(index: 0, changeType: .updated)
        } else {
            // If they are not local, add them to the array as a newly joined
 participant.
            participants.append(StageParticipant(isLocal: false, participantId:
 participant.participantId))
            participantsChanged(index: (participants.count - 1), changeType: .joined)
        }
    }

    func stage(_ stage: IVSStage, participantDidLeave participant: IVSParticipantInfo)
 {
        if participant.isLocal {
```

```
            // If this is the local participant leaving the Stage, update the first
participant in our array because
            // we want to keep the camera preview active
            participants[0].participantId = nil
            participantsChanged(index: 0, changeType: .updated)
        } else {
            // If they are not local, find their index and remove them from the array.
            if let index = participants.firstIndex(where: { $0.participantId ==
participant.participantId }) {
                participants.remove(at: index)
                participantsChanged(index: index, changeType: .left)
            }
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange
publishState: IVSParticipantPublishState) {
        // Update the publishing state of this participant
        mutatingParticipant(participant.participantId) { data in
            data.publishState = publishState
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange
subscribeState: IVSParticipantSubscribeState) {
        // Update the subscribe state of this participant
        mutatingParticipant(participant.participantId) { data in
            data.subscribeState = subscribeState
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo,
didChangeMutedStreams streams: [IVSStageStream]) {
        // We don't want to take any action for the local participant because we track
those streams locally
        if participant.isLocal { return }
        // For remote participants, notify the UICollectionView that they have updated.
There is no need to modify
        // the `streams` property on the `StageParticipant` because it is the same
`IVSStageStream` instance. Just
        // query the `isMuted` property again.
        if let index = participants.firstIndex(where: { $0.participantId ==
participant.participantId }) {
            participantsChanged(index: index, changeType: .updated)
```

```
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didAdd streams:
[IVSStageStream]) {
        // We don't want to take any action for the local participant because we track
those streams locally
        if participant.isLocal { return }
        // For remote participants, add these new streams to that participant's streams
array.
        mutatingParticipant(participant.participantId) { data in
            data.streams.append(contentsOf: streams)
        }
    }

    func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didRemove streams:
[IVSStageStream]) {
        // We don't want to take any action for the local participant because we track
those streams locally
        if participant.isLocal { return }
        // For remote participants, remove these streams from that participant's
streams array.
        mutatingParticipant(participant.participantId) { data in
            let oldUrns = streams.map { $0.device.descriptor().urn }
            data.streams.removeAll(where: { stream in
                return oldUrns.contains(stream.device.descriptor().urn)
            })
        }
    }

    // A helper function to find a participant by its ID, mutate that participant, and
then update the UICollectionView accordingly.
    private func mutatingParticipant(_ participantId: String?, modifier: (inout
StageParticipant) -> Void) {
        guard let index = participants.firstIndex(where: { $0.participantId ==
participantId }) else {
            fatalError("Something is out of sync, investigate if this was a sample app
or SDK issue.")
        }

        var participant = participants[index]
        modifier(&participant)
        participants[index] = participant
        participantsChanged(index: index, changeType: .updated)
```

```
    }
}
```

Este código usa una extensión para convertir el estado de conexión en texto legible para nosotros:

```
extension IVSStageConnectionState {
    var text: String {
        switch self {
        case .disconnected: return "Disconnected"
        case .connecting: return "Connecting"
        case .connected: return "Connected"
        @unknown default: fatalError()
        }
    }
}
```

Implementación de una interfaz de usuario personalizada CollectionViewLayout

Establecer diferentes números de participantes puede resultar complejo. Lo ideal es que ocupen todo el marco de la vista principal, pero gestionar la configuración de cada participante de forma independiente no es la mejor forma de lograrlo. Para facilitarlo, veremos cómo implementar `UICollectionViewLayout`.

Cree otro archivo, `ParticipantCollectionViewLayout.swift`, que debería ampliar `UICollectionViewLayout`. Esta clase usará otra clase llamada `StageLayoutCalculator`, que veremos pronto. La clase recibe los valores de marco calculados para cada participante y, a continuación, genera los objetos de `UICollectionViewLayoutAttributes` necesarios.

```
import Foundation
import UIKit

/**
 Code modified from https://developer.apple.com/documentation/uikit/views_and_controls/
collection_views/layouts/customizing_collection_view_layouts?language=objc
 */
class ParticipantCollectionViewLayout: UICollectionViewLayout {

    private let layoutCalculator = StageLayoutCalculator()

    private var contentBounds = CGRect.zero
    private var cachedAttributes = [UICollectionViewLayoutAttributes]()
```

```
    override func prepare() {
        super.prepare()

        guard let collectionView = collectionView else { return }

        cachedAttributes.removeAll()
        contentBounds = CGRect(origin: .zero, size: collectionView.bounds.size)

        layoutCalculator.calculateFrames(participantCount:
collectionView.numberOfItems(inSection: 0),
                                         width: collectionView.bounds.size.width,
                                         height: collectionView.bounds.size.height,
                                         padding: 4)
        .enumerated()
        .forEach { (index, frame) in
            let attributes = UICollectionViewLayoutAttributes(forCellWith:
IndexPath(item: index, section: 0))
            attributes.frame = frame
            cachedAttributes.append(attributes)
            contentBounds = contentBounds.union(frame)
        }
    }

    override var collectionViewContentSize: CGSize {
        return contentBounds.size
    }

    override func shouldInvalidateLayout(forBoundsChange newBounds: CGRect) -> Bool {
        guard let collectionView = collectionView else { return false }
        return !newBounds.size.equalTo(collectionView.bounds.size)
    }

    override func layoutAttributesForItem(at indexPath: IndexPath) ->
UICollectionViewLayoutAttributes? {
        return cachedAttributes[indexPath.item]
    }

    override func layoutAttributesForElements(in rect: CGRect) ->
[UICollectionViewLayoutAttributes]? {
        var attributesArray = [UICollectionViewLayoutAttributes]()

        // Find any cell that sits within the query rect.
```

```
        guard let lastIndex = cachedAttributes.indices.last, let firstMatchIndex =
 binSearch(rect, start: 0, end: lastIndex) else {
            return attributesArray
        }

        // Starting from the match, loop up and down through the array until all the
 attributes
        // have been added within the query rect.
        for attributes in cachedAttributes[..<firstMatchIndex].reversed() {
            guard attributes.frame.maxY >= rect.minY else { break }
            attributesArray.append(attributes)
        }

        for attributes in cachedAttributes[firstMatchIndex...] {
            guard attributes.frame.minY <= rect.maxY else { break }
            attributesArray.append(attributes)
        }

        return attributesArray
    }

    // Perform a binary search on the cached attributes array.
    func binSearch(_ rect: CGRect, start: Int, end: Int) -> Int? {
        if end < start { return nil }

        let mid = (start + end) / 2
        let attr = cachedAttributes[mid]

        if attr.frame.intersects(rect) {
            return mid
        } else {
            if attr.frame.maxY < rect.minY {
                return binSearch(rect, start: (mid + 1), end: end)
            } else {
                return binSearch(rect, start: start, end: (mid - 1))
            }
        }
    }
}
```

La clase `StageLayoutCalculator.swift` es más importante. Esto está pensado para calcular los marcos de cada participante en función del número de participantes en un diseño de filas o columnas basado en flujos. Cada fila tiene la misma altura que las demás, pero las columnas

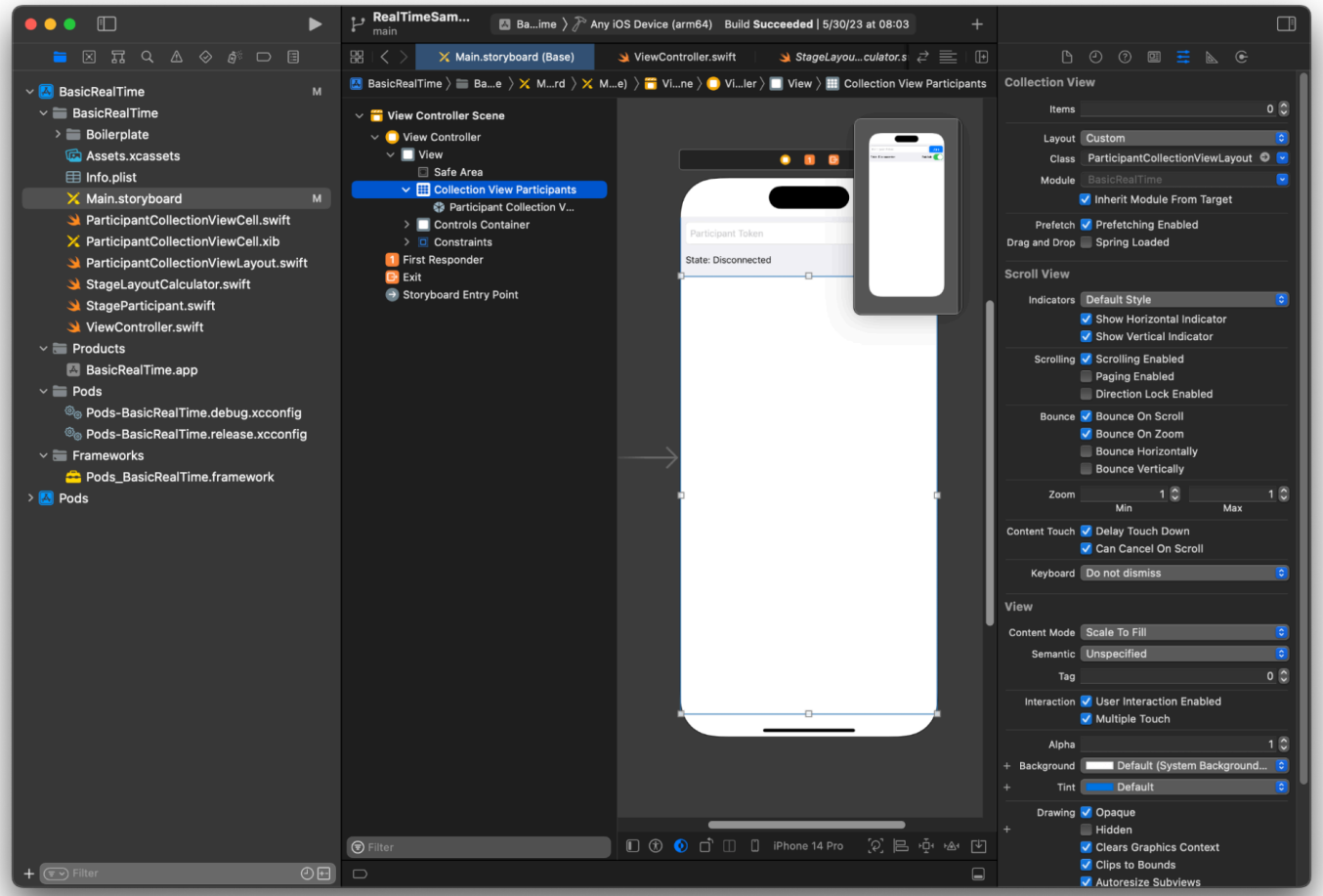
pueden tener diferentes anchuras por fila. Consulte el comentario del código que aparece sobre la variable `layouts` para ver cómo personalizar este comportamiento.

```
import Foundation
import UIKit

class StageLayoutCalculator {

    /// This 2D array contains the description of how the grid of participants should
 be rendered
    /// The index of the 1st dimension is the number of participants needed to active
 that configuration
    /// Meaning if there is 1 participant, index 0 will be used. If there are 5
 participants, index 4 will be used.
    ///
    /// The 2nd dimension is a description of the layout. The length of the array is
 the number of rows that
    /// will exist, and then each number within that array is the number of columns in
 each row.
    ///
    /// See the code comments next to each index for concrete examples.
    ///
    /// This can be customized to fit any layout configuration needed.
    private let layouts: [[Int]] = [
        // 1 participant
        [ 1 ], // 1 row, full width
        // 2 participants
        [ 1, 1 ], // 2 rows, all columns are full width
        // 3 participants
        [ 1, 2 ], // 2 rows, first row's column is full width then 2nd row's columns
 are 1/2 width
        // 4 participants
        [ 2, 2 ], // 2 rows, all columns are 1/2 width
        // 5 participants
        [ 1, 2, 2 ], // 3 rows, first row's column is full width, 2nd and 3rd row's
 columns are 1/2 width
        // 6 participants
        [ 2, 2, 2 ], // 3 rows, all column are 1/2 width
        // 7 participants
        [ 2, 2, 3 ], // 3 rows, 1st and 2nd row's columns are 1/2 width, 3rd row's
 columns are 1/3rd width
        // 8 participants
        [ 2, 3, 3 ],
```

```
        // 9 participants
        [ 3, 3, 3 ],
        // 10 participants
        [ 2, 3, 2, 3 ],
        // 11 participants
        [ 2, 3, 3, 3 ],
        // 12 participants
        [ 3, 3, 3, 3 ],
    ]

    // Given a frame (this could be for a UICollectionView, or a Broadcast Mixer's
canvas), calculate the frames for each
    // participant, with optional padding.
    func calculateFrames(participantCount: Int, width: CGFloat, height: CGFloat,
padding: CGFloat) -> [CGRect] {
        if participantCount > layouts.count {
            fatalError("Only \(layouts.count) participants are supported at this time")
        }
        if participantCount == 0 {
            return []
        }
        var currentIndex = 0
        var lastFrame: CGRect = .zero

        // If the height is less than the width, the rows and columns will be flipped.
        // Meaning for 6 participants, there will be 2 rows of 3 columns each.
        let isVertical = height > width

        let halfPadding = padding / 2.0

        let layout = layouts[participantCount - 1] // 1 participant is in index 0, so
`-1`.
        let rowHeight = (isVertical ? height : width) / CGFloat(layout.count)

        var frames = [CGRect]()
        for row in 0 ..< layout.count {
            // layout[row] is the number of columns in a layout
            let itemWidth = (isVertical ? width : height) / CGFloat(layout[row])
            let segmentFrame = CGRect(x: (isVertical ? 0 : lastFrame.maxX) +
halfPadding,
                                      y: (isVertical ? lastFrame.maxY : 0) +
halfPadding,
                                      width: (isVertical ? itemWidth : rowHeight) -
padding,
```

```
                                        height: (isVertical ? rowHeight : itemWidth) -
 padding)

            for column in 0 ..< layout[row] {
                var frame = segmentFrame
                if isVertical {
                    frame.origin.x = (itemWidth * CGFloat(column)) + halfPadding
                } else {
                    frame.origin.y = (itemWidth * CGFloat(column)) + halfPadding
                }
                frames.append(frame)
                currentIndex += 1
            }

            lastFrame = segmentFrame
            lastFrame.origin.x += halfPadding
            lastFrame.origin.y += halfPadding
        }
        return frames
    }

}
```

En `Main.storyboard`, asegúrese de establecer la clase que acabamos de crear como la clase de diseño de `UICollectionView`:



Enlace de acciones de la interfaz de usuario

Ya casi hemos terminado, pero aún tenemos que crear unas cuantas `IBActions`.

Primero, nos encargamos del botón de unión. Responde de manera diferente en función del valor de `connectingOrConnected`. Cuando está conectado, simplemente abandona el escenario. Si está desconectado, lee el texto del token `UITextField` y crea un nuevo `IVSStage` con ese texto. A continuación, agregamos `ViewController` como `strategy`, `errorDelegate` y renderizador para `IVSStage`. Finalmente, nos unimos al escenario de forma asíncrona.

```
@IBAction private func joinTapped(_ sender: UIButton) {
    if connectingOrConnected {
        // If we're already connected to a Stage, leave it.
        stage?.leave()
    } else {
        guard let token = textFieldToken.text else {
```

```
            print("No token")
            return
        }
        // Hide the keyboard after tapping Join
        textFieldToken.resignFirstResponder()
        do {
            // Destroy the old Stage first before creating a new one.
            self.stage = nil
            let stage = try IVSStage(token: token, strategy: self)
            stage.errorDelegate = self
            stage.addRenderer(self)
            try stage.join()
            self.stage = stage
        } catch {
            print("Failed to join stage - \(error)")
        }
    }
}
```

La otra acción de la interfaz de usuario que tenemos que enlazar es la opción de publicación:

```
@IBAction private func publishToggled(_ sender: UISwitch) {
    // Because the strategy returns the value of `switchPublish.isOn`, just call
 `refreshStrategy`.
    stage?.refreshStrategy()
}
```

Renderización de los participantes

Por último, tenemos que renderizar los datos que recibimos del SDK en la celda de participante que creamos anteriormente. Ya hemos completado la lógica de `UICollectionView`, por lo que solo tenemos que implementar la API `set` en `ParticipantCollectionViewCell.swift`.

Empezamos por agregar la función `empty` y, luego, la analizaremos paso a paso:

```
func set(participant: StageParticipant) {

}
```

Primero, analizamos el estado sencillo, el ID del participante, el estado de la publicación y el estado de la suscripción. Para hacerlo, tan solo actualizamos `UILabels` directamente:

```
labelParticipantId.text = participant.isLocal ? "You (\(participant.participantId ??
 "Disconnected"))" : participant.participantId
labelPublishState.text = participant.publishState.text
labelSubscribeState.text = participant.subscribeState.text
```

Las propiedades de texto de las enumeraciones de publicación y suscripción provienen de extensiones locales:

```
extension IVSParticipantPublishState {
    var text: String {
        switch self {
        case .notPublished: return "Not Published"
        case .attemptingPublish: return "Attempting to Publish"
        case .published: return "Published"
        @unknown default: fatalError()
        }
    }
}

extension IVSParticipantSubscribeState {
    var text: String {
        switch self {
        case .notSubscribed: return "Not Subscribed"
        case .attemptingSubscribe: return "Attempting to Subscribe"
        case .subscribed: return "Subscribed"
        @unknown default: fatalError()
        }
    }
}
```

A continuación, actualizamos los estados silenciados de audio y video. Para obtener los estados silenciados, tenemos que encontrar el `IVSImageDevice` y el `IVSAudioDevice` de la matriz `streams`. Para optimizar el rendimiento, recordaremos los últimos dispositivos conectados.

```
// This belongs outside `set(participant:)`
private var registeredStreams: Set<IVSStageStream> = []
private var imageDevice: IVSImageDevice? {
    return registeredStreams.lazy.compactMap { $0.device as? IVSImageDevice }.first
}
private var audioDevice: IVSAudioDevice? {
    return registeredStreams.lazy.compactMap { $0.device as? IVSAudioDevice }.first
}
```

```
// This belongs inside `set(participant:)`
let existingAudioStream = registeredStreams.first { $0.device is IVSAudioDevice }
let existingImageStream = registeredStreams.first { $0.device is IVSImageDevice }

registeredStreams = Set(participant.streams)

let newAudioStream = participant.streams.first { $0.device is IVSAudioDevice }
let newImageStream = participant.streams.first { $0.device is IVSImageDevice }

// `isMuted != false` covers the stream not existing, as well as being muted.
labelVideoMuted.text = "Video Muted: \(newImageStream?.isMuted != false)"
labelAudioMuted.text = "Audio Muted: \(newAudioStream?.isMuted != false)"
```

Por último, queremos renderizar una vista previa de `imageDevice` y mostrar las estadísticas de audio de `audioDevice`:

```
if existingImageStream !== newImageStream {
    // The image stream has changed
    updatePreview() // We'll cover this next
}

if existingAudioStream !== newAudioStream {
    (existingAudioStream?.device as? IVSAudioDevice)?.setStatsCallback(nil)
    audioDevice?.setStatsCallback( { [weak self] stats in
        self?.labelAudioVolume.text = String(format: "Audio Level: %.0f dB", stats.rms)
    })
    // When the audio stream changes, it will take some time to receive new stats.
 Reset the value temporarily.
    self.labelAudioVolume.text = "Audio Level: -100 dB"
}
```

La última función que tenemos que crear es `updatePreview()`, que agrega una vista previa del participante a nuestra vista:

```
private func updatePreview() {
    // Remove any old previews from the preview container
    viewPreviewContainer.subviews.forEach { $0.removeFromSuperview() }
    if let imageDevice = self.imageDevice {
        if let preview = try? imageDevice.previewView(with: .fit) {
            viewPreviewContainer.addSubviewMatchFrame(preview)
        }
```

```
    }
}
```

Lo anterior usa una función auxiliar en `UIView` para facilitar la incorporación de subvistas:

```
extension UIView {
    func addSubviewMatchFrame(_ view: UIView) {
        view.translatesAutoresizingMaskIntoConstraints = false
        self.addSubview(view)
        NSLayoutConstraint.activate([
            view.topAnchor.constraint(equalTo: self.topAnchor, constant: 0),
            view.bottomAnchor.constraint(equalTo: self.bottomAnchor, constant: 0),
            view.leadingAnchor.constraint(equalTo: self.leadingAnchor, constant: 0),
            view.trailingAnchor.constraint(equalTo: self.trailingAnchor, constant: 0),
        ])
    }
}
```

Supervisión del streaming en tiempo real de Amazon IVS

¿Qué es la sesión de una fase?

La sesión de una fase comienza cuando el primer participante se une a una fase y finaliza unos minutos después de que el último participante deje de publicar en dicha fase. Las sesiones de las fases ayudan a depurar las fases de larga duración al separar los eventos y los participantes en sesiones de corta duración.

Ver sesiones de fases y participantes

Instrucciones de la consola

1. Abra la consola de Amazon IVS.

   (También puede acceder a la consola de Amazon IVS a través de la Consola de administración de AWS).
2. En el panel de navegación elija Fases. (Si el panel de navegación está contraído, ábralo primero eligiendo el icono de hamburguesa).
3. Elija la fase para ir a la página de detalles correspondiente.
4. Desplácese hacia abajo en la página hasta que vea la sección Sesiones de fase y, a continuación, seleccione la sesión de una fase para ver la página de detalles.
5. Para ver los participantes de la sesión, desplácese hacia abajo hasta que vea la sección Participantes y, a continuación, seleccione un participante para ver la página de detalles pertinente, incluidos los gráficos de las métricas de Amazon CloudWatch.

Ver eventos de un participante

Los eventos se envían cuando el estado de un participante en de un escenario cambia. Por ejemplo, cuando se une a un escenario o se produce un error al intentar publicar en uno de ellos. No todos los errores provocan eventos; por ejemplo, los errores de red del lado del cliente y los errores de firma de los tokens no se envían como eventos. Para gestionar estos errores en la aplicación cliente, utilice los SDK de transmisión de IVS.

Instrucciones de la consola

1. Vaya a la página de detalles del participante tal y como se ha indicado anteriormente.
2. Desplácese hacia abajo hasta que vea la sección Eventos. Se muestra una lista ordenada de los eventos del participante. Consulte Uso de Amazon EventBridge con Amazon IVS para obtener más información sobre los eventos que se emiten para los participantes.

Instrucciones de la CLI

El acceso a los eventos de sesiones de escenarios con la AWS CLI es una opción avanzada y requiere que antes descargue y configure la CLI en su equipo. Para obtener más información, consulte la Guía del usuario de la Interfaz de la línea de comandos de AWS.

1. Enumere las sesiones de una fase para encontrar una en particular:

```
aws ivs-realtime list-stage-sessions --stage-arn <arn>
```

2. Enumere los participantes de la sesión de una fase para encontrar un participante:

```
aws ivs-realtime list-participants --stage-arn <arn> –session-id <sessionId>
```

3. Enumere los eventos de la sesión de una fase y un participante:

```
aws ivs-realtime list-participant-events --stage-arn <arn> --session-id <sessionId>
 --participant-id <participantId>
```

A continuación, se muestra una respuesta de ejemplo a la llamada `list-participant-events`:

```
{
    "events": [
        {
            "eventTime": "2023-04-04T22:48:41+00:00",
            "name": "JOINED",
            "participantId": "AdRezBl021t0"
        },
        {
            "eventTime": "2023-04-04T22:48:41+00:00",
            "name": "SUBSCRIBE_STARTED",
            "participantId": "AdRezBl021t0",
```

```
            "remoteParticipantId": "Ou5b5n5XLMdC"
        },
        {
            "eventTime": "2023-04-04T22:49:45+00:00",
            "name": "SUBSCRIBE_STOPPED",
            "participantId": "AdRezBl021t0",
            "remoteParticipantId": "Ou5b5n5XLMdC"
        },
        {
            "eventTime": "2023-04-04T22:49:45+00:00",
            "name": "LEFT",
            "participantId": "AdRezBl021t0"
        }
    ]
}
```

Acceso a métricas de CloudWatch

Para que las métricas de CloudWatch estén disponibles, se requieren las siguientes versiones del SDK de transmisión de IVS: Web 1.5.0 o posterior, Android 1.12.0 o posterior o iOS 1.12.0 o posterior.

Instrucciones de la consola de CloudWatch

1. Abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
2. En el panel de navegación lateral, expanda el menú desplegable Metrics (Métricas) y, a continuación, seleccione All metrics (Todas las métricas).
3. En la pestaña Explorar, mediante el menú desplegable sin etiqueta de la izquierda, seleccione su región de “inicio”, donde se crearon los canales. Para obtener más información sobre las regiones, consulte Solución global, control regional. Para obtener una lista de las regiones admitidas, consulte la Página de Amazon IVS en la Referencia general de AWS.
4. En la parte inferior de la pestaña Explorar, seleccione el espacio de nombres IVSRealTime.
5. Haga una de las acciones siguientes:
   a. En la barra de búsqueda, ingrese el ID de recurso (parte del ARN, `arn:::ivs:stage/<resource id>`).

      A continuación, seleccione IVSRealTime > Estado de las métricas.

b. Si IVSRealTime aparece como un servicio seleccionable en Espacios de nombres de AWS, selecciónelo. Se mostrará si utiliza transmisión en tiempo real de Amazon IVS y envía métricas a Amazon CloudWatch. (Si IVSRealTime no está en la lista, no tiene ninguna métrica de Amazon IVS).

   Luego, elija la agrupación de dimensiones que desee; las dimensiones disponibles se muestran a continuación en Métricas de CloudWatch.

6. Elija métricas para agregarlas al gráfico. Las métricas disponibles se muestran a continuación en Métricas de CloudWatch.

También puede acceder al gráfico de CloudWatch de la sesión de transmisión desde la página de detalles de la sesión de transmisión, seleccionando el botón View in CloudWatch (Ver en CloudWatch).

Instrucciones de la CLI

También puede obtener acceso a las métricas mediante la AWS CLI. Esto requiere que primero descargue y configure la CLI en su equipo. Para obtener más información, consulte la Guía del usuario de la interfaz de línea de comandos de AWS.

A continuación, para obtener acceso a las métricas de transmisión en tiempo real de Amazon IVS mediante AWS CLI:

- En el símbolo del sistema, ejecute:

```
aws cloudwatch list-metrics --namespace AWS/IVSRealTime
```

Para obtener más información, consulte Uso de las métricas de Amazon CloudWatch en la Guía del usuario de Amazon CloudWatch.

Métricas de CloudWatch: streaming en tiempo real de IVS

Amazon IVS proporciona las siguientes métricas en el espacio de nombres de AWS/IVSRealTime.

Para que las métricas de CloudWatch estén disponibles, se debe usar SDK de transmisión web 1.5.2 o una versión posterior.

La dimensión puede tener los siguientes valores válidos:

- La dimensión `Stage` es un ID de recurso (parte del ARN, `arn:::stage/<resource id>`).
- La dimensión de `Participant` es una `participantID`.
- `SimulcastLayer` es “alta”, “media”, “baja” o “sin RID” para el `MediaType` “video” o “deshabilitada” para el `MediaType` “audio” Este valor también puede estar vacío.
- La dimensión de `MediaType` es “video” o “audio” (cadena).

Métrica	Dimensión	Descripción
`DownloadPacketLoss`	`Stage`	Cada muestra representa el porcentaje de paquetes que perdió un suscriptor determinado durante la descarga del servidor de IVS. Unidad: porcentaje Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) pérdida de paquetes durante el intervalo configurado
`DownloadPacketLoss`	`Stage,Participant`	Filtra `DownloadPacketLoss` por participante, para los suscriptores que también son publicadores. Las muestras representan el porcentaje de paquetes que perdió un suscriptor durante la descarga del servidor de IVS. Las muestras se emiten solo cuando el participante también es publicador. Unidad: porcentaje Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) de fotogramas descartados durante el intervalo configurado
`DroppedFrames`	`Stage`	Cada muestra representa el porcentaje de fotogramas descartados por un suscriptor determinado. Unidad: porcentaje

Métrica	Dimensión	Descripción
		Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) de fotogramas descartados durante el intervalo configurado
`DroppedFrames`	`Stage,Participant`	Filtra `DroppedFrames` por participante, para los suscriptores que también son publicadores. Los ejemplos representan el porcentaje de fotogramas que se eliminaron entre el participante suscriptor y todos los editores del escenario. Las muestras se emiten solo cuando el participante también es publicador. Unidad: porcentaje Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) de fotogramas descartados durante el intervalo configurado
`PublishBitrate`	`Stage`	Las muestras emitidas representan la velocidad total a la que un publicador envía datos de video y audio (sumados en todas las capas de transmisión simultánea). Unidad: bits por segundo Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) de tasas de bits durante el intervalo configurado

Métrica	Dimensión	Descripción
`PublishBitrate`	`Stage, Participant, SimulcastLayer, MediaType`	Filtra `PublishBitrate` por participante, capa de transmisión simultánea y tipo de medio. El ID de la capa de transmisión simultánea lo establece el SDK de transmisión. Cuando la transmisión simultánea está deshabilitada, este ID de capa se establece como “deshabilitado”. El tipo de medio es video o audio. Unidad: bits por segundo Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) de tasas de bits durante el intervalo configurado
`Publishers`	`Stage`	Número de participantes que publican en el escenario. Unidad: recuento Estadísticas válidas: Promedio, Máximo, Mínimo
`PublishResolution`	`Stage, Participant, SimulcastLayer, MediaType`	Número de píxeles en el ancho o alto más pequeño del marco. Por ejemplo, para un marco horizontal de 1920 x 1080, la PublishResolution es 1080. Para un marco de retrato de tamaño 720 x 1280, la PublishResolution es 720. Unidad: recuento Estadísticas válidas: Promedio, Máximo, Mínimo
`SubscribeBitrate`	`Stage`	Las muestras emitidas representan la velocidad total a la que un suscriptor determinado recibe datos de video y audio. Unidad: bits por segundo Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) de tasas de bits durante el intervalo configurado

Métrica	Dimensión	Descripción
`SubscribeBitrate`	`Stage,Participant,MediaType`	Filtra `SubscribeBitrate` por participante, para los suscriptores que también son publicadores. Las muestras representan la velocidad de bits a la que un suscriptor determinado recibe el `MediaType` determinado. Las muestras solo se emiten mientras el participante suscriptor también es publicador. Unidad: bits por segundo Estadísticas válidas: promedio, máximo, mínimo: número medio, número mayor o menor (respectivamente) de tasas de bits durante el intervalo configurado
`Subscribers`	`Stage`	Número de participantes suscritos al escenario. Tenga en cuenta que los participantes que publican y se suscriben activamente se cuentan tanto como editores como suscriptores. Unidad: recuento Estadísticas válidas: Promedio, Máximo, Mínimo

SDK de transmisión de IVS (streaming en tiempo real)

El SDK de transmisión para streaming en tiempo real de Amazon Interactive Video Service (IVS) está pensado para desarrolladores que crean aplicaciones con Amazon IVS. Este SDK está diseñado a fin de aprovechar la arquitectura de Amazon IVS y verá mejoras continuas y nuevas características, junto con Amazon IVS. Como SDK de transmisión nativo, está diseñado para minimizar el impacto en el rendimiento de la aplicación y en los dispositivos con los que los usuarios acceden a la aplicación.

Tenga en cuenta que el SDK de transmisión se usa tanto para enviar como para recibir videos; es decir, utiliza el mismo SDK para los hosts y para los espectadores. No se necesita el SDK de un reproductor independiente.

Su aplicación puede aprovechar las características clave del SDK de transmisión de Amazon IVS:

- Streaming de alta calidad: el SDK de transmisión admite el streaming de alta calidad. Reciba video de su cámara y codifíquelo a una velocidad de hasta 720p.
- Ajustes automáticos de la velocidad de bits: los usuarios de smartphones son móviles, por lo que sus condiciones de red pueden cambiar a lo largo de una transmisión. El SDK de transmisión de Amazon IVS ajusta automáticamente la velocidad de bits de video para adaptarse a las condiciones cambiantes de la red.
- Soporte vertical y horizontal: independientemente del modo en que los usuarios mantengan sus dispositivos, la imagen aparece con el lado correcto hacia arriba y se escala según corresponda. El SDK de transmisión admite el tamaño de formato vertical y horizontal. Administra automáticamente la relación de aspecto cuando los usuarios rotan su dispositivo hacia una orientación distinta de la configurada.
- Streaming seguro: las transmisiones de su usuario se cifran mediante TLS, por lo que pueden mantener sus transmisiones seguras.
- Dispositivos de audio externos: el SDK de transmisión de Amazon IVS admite micrófonos externos SCO de audio, USB y Bluetooth.

Requisitos de la plataforma

Plataformas nativas

Plataforma	Versiones compatibles
Android	9.0 y versiones posteriores: tenga en cuenta que los clientes pueden crear con la versión 5.0, pero no podrán utilizar la funcionalidad de streaming en tiempo real.
iOS	14 y versiones posteriores

IVS admite un mínimo de 4 versiones principales de iOS y 6 versiones principales de Android. El soporte de nuestra versión actual puede extenderse más allá de estos mínimos. Los clientes recibirán una notificación mediante las notas de lanzamiento del SDK con al menos 3 meses de antelación cuando una versión principal deje de ser compatible.

Navegadores de escritorio

Navegador	Plataformas admitidas	Versiones compatibles
Chrome	Windows, macOS	Dos versiones principales (la versión actual y la anterior más reciente)
Firefox	Windows, macOS	Dos versiones principales (la versión actual y la anterior más reciente)
Ubicaciones	Windows 8.1 y posteriores	Dos versiones principales (la versión actual y la anterior más reciente) No incluye Edge Legacy
Safari	macOS	Dos versiones principales (la versión actual y la anterior más reciente)

Navegadores móviles (iOS y Android)

Navegador	Plataformas admitidas	Versiones compatibles
Chrome	iOS, Android	Dos versiones principales (la versión actual y la anterior más reciente)
Firefox	Android	Dos versiones principales (la versión actual y la anterior más reciente)
Safari	iOS	Dos versiones principales (la versión actual y la anterior más reciente)

Limitaciones conocidas

- En todos los dispositivos móviles, no recomendamos publicar o suscribirse con cuatro o más participantes al mismo tiempo, debido a problemas con los artefactos de vídeo y las pantallas negras. Si necesita más participantes, configure la opción de publicar y suscribirse solo en audio.
- No recomendamos componer un escenario y retransmitirlo a un canal de Android Mobile Web por motivos de rendimiento y posibles bloqueos. Si se requiere la funcionalidad de transmisión, integre el SDK de transmisión para streaming en tiempo real de IVS para Android.

Vistas web

El SDK de transmisión web no admite vistas web ni entornos similares a los de la web (televisores, consolas, etc.). Para implementaciones móviles, consulte la Guía del SDK de transmisión de streaming en tiempo para Android y para iOS.

Se requiere acceso a los dispositivos

El SDK de difusión requiere acceso a las cámaras y micrófonos del dispositivo, tanto los integrados en el dispositivo como los conectados a través de Bluetooth, USB o conector de audio.

Soporte

El SDK de transmisión se mejora de forma continua. Consulte Notas de la versión de Amazon IVS para ver las versiones disponibles y los problemas solucionados. Si procede, antes de contactar con el soporte técnico, actualice su versión del SDK de transmisión y compruebe si se resuelve el problema.

Control de versiones

Los SDK de transmisión de Amazon IVS utilizan el control de versiones semántico.

Para este análisis, suponga:

- La última versión es la 4.1.3.
- La última versión de la versión principal anterior es la 3.2.4.
- La última versión de la versión 1.x es la 1.5.6.

Las características nuevas compatibles con versiones anteriores se agregan como versiones secundarias de la última versión. En este caso, el siguiente conjunto de características nuevas se agregará como la versión 4.2.0.

Se agregan correcciones de errores menores compatibles con versiones anteriores como parches de la última versión. Aquí, el siguiente conjunto de correcciones de errores menores se agregará como la versión 4.1.4.

Las correcciones de errores principales compatibles con versiones anteriores se manejan de manera diferente; estas se agregan a varias versiones:

- Versión del parche de la última versión. Aquí, esta es la versión 4.1.4.
- Versión del parche de la versión secundaria anterior. Aquí, esta es la versión 3.2.5.
- Versión del parche de la última versión 1.x. Aquí, esta es la versión 1.5.7.

El equipo de productos de Amazon IVS define las principales correcciones de errores. Las actualizaciones de seguridad críticas y otras correcciones seleccionadas necesarias para los clientes son ejemplos típicos.

Nota: En los ejemplos anteriores, las versiones publicadas aumentan sin omitir ningún número (por ejemplo, de 4.1.3 a 4.1.4). En realidad, uno o más números de parche pueden permanecer internos y no ser lanzados, por lo que la versión publicada podría aumentar de 4.1.3 a 4.1.6.

SDK de transmisión de IVS: guía para web (transmisión en tiempo real)

El SDK de transmisión web para transmisión en tiempo real de IVS ofrece a los desarrolladores las herramientas para crear experiencias interactivas y en tiempo real en la web. El SDK está pensado para los desarrolladores que crean aplicaciones web con Amazon IVS.

El SDK de transmisión web permite a los participantes enviar y recibir videos. El SDK admite las siguientes operaciones:

- Incorporación a un escenario
- Publicación de contenido multimedia para otros participantes del escenario
- Suscripción al contenido multimedia de otros participantes del escenario
- Administración y monitoreo del video y audio publicados en el escenario
- Obtención de estadísticas de WebRTC de cada conexión de pares
- Todas las operaciones del SDK de transmisión web de transmisión de baja latencia de IVS

Última versión del SDK de transmisión web: 1.8.0 (notas de la versión)

Documentación de referencia: para obtener información sobre los métodos más importantes disponibles en el SDK de Amazon IVS Web Broadcast, consulte https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference. Asegúrese de seleccionar la versión más reciente del SDK.

Código de ejemplo: los siguientes ejemplos son un buen punto para empezar rápidamente a usar el SDK:

- HTML y JavaScript
- React

Requisitos de la plataforma: consulte SDK de transmisión de Amazon IVS para obtener un listado de las plataformas compatibles.

Introducción

Importaciones

Los componentes básicos de la transmisión en tiempo real se encuentran en un espacio de nombres diferente al de los módulos de transmisión raíz.

Uso de una etiqueta de script

Con las mismas importaciones de scripts, las clases y enumeraciones definidas en los ejemplos siguientes se pueden encontrar en el objeto global `IVSBroadcastClient`:

```
const { Stage, SubscribeType } = IVSBroadcastClient;
```

Con npm

Las clases, enumeraciones y tipos también se pueden importar desde el módulo del paquete:

```
import { Stage, SubscribeType, LocalStageStream } from 'amazon-ivs-web-broadcast'
```

Solicitar permisos

La aplicación debe solicitar permiso para acceder a la cámara y al micrófono del usuario, y deberán ser ofrecidos mediante HTTPS. (Esto no es específico de Amazon IVS; es necesario para cualquier sitio web que necesite acceso a cámaras y micrófonos).

A continuación, le presentamos una función a modo de ejemplo que muestra cómo solicitar y capturar permisos para los dispositivos de audio y video:

```
async function handlePermissions() {
   let permissions = {
       audio: false,
       video: false,
   };
   try {
       const stream = await navigator.mediaDevices.getUserMedia({ video: true, audio:
 true });
       for (const track of stream.getTracks()) {
           track.stop();
```

```
        }
        permissions = { video: true, audio: true };
    } catch (err) {
        permissions = { video: false, audio: false };
        console.error(err.message);
    }
    // If we still don't have permissions after requesting them display the error
 message
    if (!permissions.video) {
        console.error('Failed to get video permissions.');
    } else if (!permissions.audio) {
        console.error('Failed to get audio permissions.');
    }
}
```

Para obtener información adicional, consulte la API de permisos y MediaDevices. getUserMedia().

Enlistar los dispositivos disponibles

Para ver qué dispositivos están disponibles para capturar, consulta el método MediaDevices.enumerateDevices () del navegador:

```
const devices = await navigator.mediaDevices.enumerateDevices();
window.videoDevices = devices.filter((d) => d.kind === 'videoinput');
window.audioDevices = devices.filter((d) => d.kind === 'audioinput');
```

Recupera un de un dispositivo MediaStream

Después de obtener la lista de dispositivos disponibles, puede recuperar una transmisión de diversos dispositivos. Por ejemplo, puede utilizar el método `getUserMedia()` para recuperar la transmisión de una cámara.

Si quisiera indicar desde cuál dispositivo desea capturar la transmisión, puede establecer de forma expresa `deviceId` en las secciones `audio` o `video` de las restricciones de multimedia. De forma alternativa, puede omitir `deviceId` y que los usuarios seleccionen los dispositivos desde el símbolo del navegador.

También puede especificar la resolución de cámara ideal mediante las restricciones `width` y `height`. (Obtenga más información sobre estas restricciones aquí). El SDK aplica de forma automática las restricciones máximas de ancho y alto que corresponden a la resolución máxima

de transmisión; sin embargo, es aconsejable que las aplique usted mismo para garantizar que la relación de aspecto de la fuente no cambie después de que la agregue al SDK.

Para la transmisión en tiempo real, asegúrese de que el contenido multimedia esté limitado a una resolución de 720p. En concreto, sus valores de ancho `getUserMedia` y alto y los de `getDisplayMedia` restricción no deben superar 921600 (1280*720) cuando se multiplican entre sí.

```
const videoConfiguration = {
  maxWidth: 1280,
  maxHeight: 720,
  maxFramerate: 30,
}

window.cameraStream = await navigator.mediaDevices.getUserMedia({
   video: {
       deviceId: window.videoDevices[0].deviceId,
       width: {
           ideal: videoConfiguration.maxWidth,
       },
       height: {
           ideal:videoConfiguration.maxHeight,
       },
   },
});
window.microphoneStream = await navigator.mediaDevices.getUserMedia({
   audio: { deviceId: window.audioDevices[0].deviceId },
});
```

Publicación y suscripción

Conceptos

Los siguientes tres conceptos básicos subyacen a la funcionalidad de transmisión en tiempo real: escenario, estrategia y eventos. El objetivo del diseño es minimizar la cantidad de lógica necesaria por parte del cliente para crear un producto que funcione.

Escenario

La clase `Stage` es el principal punto de interacción entre la aplicación host y el SDK. Representa el escenario como tal y se usa para entrar y salir de él. Para crear un escenario e incorporarse a él, es necesaria una cadena de símbolos válida y que no haya vencido del plano de control (representada como `token`). Entrar y salir de un escenario es sencillo:

```
const stage = new Stage(token, strategy)

try {
   await stage.join();
} catch (error) {
   // handle join exception
}

stage.leave();
```

Strategy (Estrategia)

La interfaz `StageStrategy` proporciona una forma para que la aplicación host comunique el estado deseado del escenario al SDK. Es necesario implementar las siguientes tres funciones: `shouldSubscribeToParticipant`, `shouldPublishParticipant` y `stageStreamsToPublish`. Todas se analizan a continuación.

Para usar una estrategia definida, pásela al constructor de `Stage`. El siguiente es un ejemplo completo de una aplicación que utiliza una estrategia para publicar la cámara web de un participante en el escenario y suscribirse a todos los participantes. El propósito de cada función de estrategia necesaria se explica detalladamente en las siguientes secciones.

```
const devices = await navigator.mediaDevices.getUserMedia({
   audio: true,
   video: {
        width: { max: 1280 },
        height: { max: 720 },
    }
});
const myAudioTrack = new LocalStageStream(devices.getAudioTracks()[0]);
const myVideoTrack = new LocalStageStream(devices.getVideoTracks()[0]);

// Define the stage strategy, implementing required functions
const strategy = {
   audioTrack: myAudioTrack,
   videoTrack: myVideoTrack,

   // optional
   updateTracks(newAudioTrack, newVideoTrack) {
      this.audioTrack = newAudioTrack;
      this.videoTrack = newVideoTrack;
   },
```

```
   // required
   stageStreamsToPublish() {
      return [this.audioTrack, this.videoTrack];
   },

   // required
   shouldPublishParticipant(participant) {
      return true;
   },

   // required
   shouldSubscribeToParticipant(participant) {
      return SubscribeType.AUDIO_VIDEO;
   }
};

// Initialize the stage and start publishing
const stage = new Stage(token, strategy);
await stage.join();


// To update later (e.g. in an onClick event handler)
strategy.updateTracks(myNewAudioTrack, myNewVideoTrack);
stage.refreshStrategy();
```

Suscripción a participantes

```
shouldSubscribeToParticipant(participant: StageParticipantInfo): SubscribeType
```

Cuando un participante remoto se incorpora al escenario, el SDK consulta la aplicación host sobre el estado de suscripción deseado de ese participante. Las opciones son `NONE`, `AUDIO_ONLY` y `AUDIO_VIDEO`. Al devolver un valor para esta función, la aplicación host no tiene que preocuparse por el estado de la publicación, el estado actual de la suscripción ni el estado de la conexión del escenario. Si se devuelve `AUDIO_VIDEO`, el SDK espera a que el participante remoto publique antes de suscribirse y actualiza la aplicación host al emitir eventos durante todo el proceso.

Este es un ejemplo de implementación:

```
const strategy = {

   shouldSubscribeToParticipant: (participant) => {
```

```
      return SubscribeType.AUDIO_VIDEO;
   }

   // ... other strategy functions
}
```

Esta es la implementación completa de esta función para una aplicación host que siempre quiere que todos los participantes se vean entre sí; por ejemplo, una aplicación de videochat.

También son posibles implementaciones más avanzadas. Utilice la propiedad `userInfo` en `ParticipantInfo` para suscribirse de forma selectiva a los participantes en función de los atributos proporcionados por el servidor:

```
const strategy = {

   shouldSubscribeToParticipant(participant) {
      switch (participant.info.userInfo) {
         case 'moderator':
            return SubscribeType.NONE;
         case 'guest':
            return SubscribeType.AUDIO_VIDEO;
         default:
            return SubscribeType.NONE;
      }
   }
   // . . . other strategies properties
}
```

Esto se puede utilizar para crear un escenario en el que los moderadores puedan monitorear a todos los invitados sin que ellos mismos los vean ni los escuchen. La aplicación host podría utilizar una lógica empresarial adicional para permitir que los moderadores se vean entre sí, pero permanezcan invisibles para los invitados.

Publicación

```
shouldPublishParticipant(participant: StageParticipantInfo): boolean
```

Una vez realizada la conexión al escenario, el SDK consulta la aplicación host para ver si un participante en particular tiene que publicar. Esto solo se invoca en los participantes locales que tienen permiso para publicar en función del token proporcionado.

Este es un ejemplo de implementación:

```
const strategy = {

   shouldPublishParticipant: (participant) => {
      return true;
   }

   // . . . other strategies properties
}
```

Esto es para una aplicación de videochat estándar en la que los usuarios siempre quieren publicar. Pueden activar y desactivar el sonido y el video para ocultarse o verse y escucharse al instante. (También pueden usar la opción de publicar o anular la publicación, pero esto es mucho más lento. Es preferible silenciar o activar el sonido en casos de uso en los que se quiera cambiar la visibilidad de manera frecuente).

Elección de las transmisiones que publicar

```
stageStreamsToPublish(): LocalStageStream[];
```

Al publicar, esto se utiliza para determinar qué transmisiones de audio y video se tienen que publicar. Esto se explica en mayor detalle más adelante en Publicación de una transmisión multimedia.

Actualización de la estrategia

La estrategia pretende ser dinámica: los valores devueltos por cualquiera de las funciones anteriores se pueden cambiar en cualquier momento. Por ejemplo, si la aplicación host no quiere publicar hasta que el usuario final presione un botón, puede devolver una variable de `shouldPublishParticipant` (como `hasUserTappedPublishButton`). Cuando esa variable cambie en función de una interacción del usuario final, llame a `stage.refreshStrategy()` para indicar al SDK que debe consultar la estrategia a fin de obtener los valores más recientes y aplicar solo los cambios. Si el SDK observa que el valor `shouldPublishParticipant` cambió, se iniciará el proceso de publicación. Si las consultas del SDK y todas las funciones devuelven el mismo valor que antes, la llamada a `refreshStrategy` no hará cambios en el escenario.

Si el valor devuelto de `shouldSubscribeToParticipant` cambia de `AUDIO_VIDEO` a `AUDIO_ONLY`, la transmisión de video se elimina para todos los participantes con valores devueltos modificados, si ya existía con anterioridad una transmisión de video.

Por lo general, el escenario utiliza la estrategia para aplicar de la manera más eficiente la diferencia entre las estrategias anteriores y actuales, sin que la aplicación host tenga que preocuparse por todo el estado necesario para administrarla correctamente. Por eso, piense en la llamada a `stage.refreshStrategy()` como una operación barata, porque no hace nada a no ser que cambie la estrategia.

Eventos

Una instancia `Stage` es un emisor de eventos. Con `stage.on()`, el estado del escenario se comunica a la aplicación host. Por lo general, los eventos permiten actualizar por completo la interfaz de usuario de la aplicación host. Los eventos son los siguientes:

```
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED, (participant, state) =>
 {})
stage.on(StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED, (participant, state) =>
 {})
stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {})
stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_REMOVED, (participant, streams) => {})
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {})
```

Para la mayoría de estos métodos, se proporciona la `ParticipantInfo` correspondiente.

No se espera que la información que proporcionan los eventos afecte a los valores de retorno de la estrategia. Por ejemplo, no se espera que el valor devuelto de `shouldSubscribeToParticipant` cambie cuando se llama a `STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`. Si la aplicación host quiere suscribirse a un participante en particular, debe devolver el tipo de suscripción deseado, independientemente del estado de publicación de ese participante. El SDK es responsable de garantizar que se aplique el estado deseado de la estrategia en el momento correcto según el estado del escenario.

Publicación de una transmisión multimedia

Los dispositivos locales, como los micrófonos y las cámaras, se recuperan siguiendo los mismos pasos descritos anteriormente en Recuperar un archivo de un dispositivo. MediaStream En el ejemplo, utilizamos `MediaStream` para crear una lista de objetos `LocalStageStream` que el SDK utiliza para publicar:

```
try {
```

```
    // Get stream using steps outlined in document above
    const stream = await getMediaStreamFromDevice();

    let streamsToPublish = stream.getTracks().map(track => {
        new LocalStageStream(track)
    });

    // Create stage with strategy, or update existing strategy
    const strategy = {
        stageStreamsToPublish: () => streamsToPublish
    }
}
```

Publicación de una pantalla compartida

Las aplicaciones suelen necesitar publicar una pantalla compartida además de la cámara web del usuario. Para publicar una pantalla compartida, es necesario crear un `Stage` adicional con su propio token único.

```
// Invoke the following lines to get the screenshare's tracks
const media = await navigator.mediaDevices.getDisplayMedia({
   video: {
      width: {
         max: 1280,
      },
      height: {
         max: 720,
      }
   }
});
const screenshare = { videoStream: new LocalStageStream(media.getVideoTracks()[0]) };
const screenshareStrategy = {
   stageStreamsToPublish: () => {
      return [screenshare.videoStream];
   },
   shouldPublishParticipant: (participant) => {
      return true;
   },
   shouldSubscribeToParticipant: (participant) => {
      return SubscribeType.AUDIO_VIDEO;
   }
}
const screenshareStage = new Stage(screenshareToken, screenshareStrategy);
```

```
await screenshareStage.join();
```

Visualización y eliminación de participantes

Cuando se complete la suscripción, recibirá una matriz de objetos `StageStream` a través del evento `STAGE_PARTICIPANT_STREAMS_ADDED`. El evento también le brinda información sobre los participantes que le será de ayuda al visualizar las transmisiones multimedia:

```
stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {
    const streamsToDisplay = streams;

    if (participant.isLocal) {
        // Ensure to exclude local audio streams, otherwise echo will occur
        streamsToDisplay = streams.filter(stream => stream.streamType ===
 StreamType.VIDEO)
    }

    // Create or find video element already available in your application
    const videoEl = getParticipantVideoElement(participant.id);

    // Attach the participants streams
    videoEl.srcObject = new MediaStream();
    streamsToDisplay.forEach(stream =>
 videoEl.srcObject.addTrack(stream.mediaStreamTrack));
})
```

Cuando un participante deja de publicar o anula la suscripción de una transmisión, se invoca la función `STAGE_PARTICIPANT_STREAMS_REMOVED` con las transmisiones que se eliminaron. Las aplicaciones host tienen que usar esto como una señal para eliminar la transmisión de video del participante de DOM.

`STAGE_PARTICIPANT_STREAMS_REMOVED` se invoca para todas las situaciones en las que se puede eliminar una transmisión, como las siguientes:

- El participante remoto deja de publicar.
- Un dispositivo local cancela la suscripción o cambia la suscripción de `AUDIO_VIDEO` a `AUDIO_ONLY`.
- El participante remoto abandona el escenario.
- El participante local abandona el escenario.

Ya que STAGE_PARTICIPANT_STREAMS_REMOVED se invoca en todas las situaciones, no es necesaria una lógica empresarial personalizada para eliminar a los participantes de la interfaz de usuario durante las operaciones de licencia remota o local.

Activación y desactivación del sonido de las transmisiones multimedia

Los objetos de LocalStageStream tienen una función setMuted que controla si la transmisión está silenciada. Esta función se puede invocar en la transmisión antes o después de que la devuelva la función de estrategia stageStreamsToPublish.

Importante: Si stageStreamsToPublish devuelve una nueva instancia de objeto de LocalStageStream después de una llamada a refreshStrategy, el estado de sonido desactivado del nuevo objeto de transmisión se aplicará al escenario. Tenga cuidado al crear nuevas instancias LocalStageStream para asegurarse de que se mantenga el estado de sonido desactivado que se espera.

Monitoreo del estado de sonido desactivado en los contenidos multimedia del participante remoto

Cuando los participantes cambian el estado de sonido desactivado de su video o audio, el evento STAGE_STREAM_MUTE_CHANGED se activa con una lista de las transmisiones que cambiaron. Use la propiedad isMuted en StageStream para actualizar su interfaz de usuario según corresponda:

```
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {
   if (stream.streamType === 'video' && stream.isMuted) {
       // handle UI changes for video track getting muted
   }
})
```

Además, puedes consultar la información sobre StageParticipantInfoel estado de silencio del audio o el vídeo:

```
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {
   if (participant.videoStopped || participant.audioMuted) {
       // handle UI changes for either video or audio
   }
})
```

Obtención de estadísticas de WebRTC

Para obtener las estadísticas más recientes de WebRTC de una transmisión de publicación o de suscripción, utilice `getStats` en `StageStream`. Este es un método asíncrono con el que puede recuperar estadísticas mediante await o encadenando una promesa. El resultado es un `RTCStatsReport`, un diccionario que contiene todas las estadísticas estándar.

```
try {
   const stats = await stream.getStats();
} catch (error) {
   // Unable to retrieve stats
}
```

Optimización del contenido multimedia

Se recomienda limitar las llamadas a `getUserMedia` y a `getDisplayMedia` con las siguientes restricciones para obtener el mejor rendimiento:

```
const CONSTRAINTS = {
    video: {
        width: { ideal: 1280 }, // Note: flip width and height values if portrait is
 desired
        height: { ideal: 720 },
        framerate: { ideal: 30 },
    },
};
```

Puede restringir aún más el contenido multimedia mediante opciones adicionales que se pasan al constructor `LocalStageStream`:

```
const localStreamOptions = {
    minBitrate?: number;
    maxBitrate?: number;
    maxFramerate?: number;
    simulcast: {
        enabled: boolean
    }
}
const localStream = new LocalStageStream(track, localStreamOptions)
```

En el código anterior:

- `minBitrate` establece la velocidad de bits mínima que se espera que utilice el navegador. Sin embargo, una transmisión de video de poca complejidad puede hacer que el codificador establezca una velocidad inferior a esta velocidad de bits.
- `maxBitrate` establece una velocidad de bits máxima que se espera que el navegador no supere para esta transmisión.
- `maxFramerate` establece una velocidad de fotogramas máxima que se espera que el navegador no supere para esta transmisión.
- La opción `simulcast` solo se puede utilizar en navegadores basados en Chromium. Permite enviar tres capas de representación de la transmisión.
  - Esto permite al servidor elegir qué representación enviar a otros participantes, en función de sus limitaciones de red.
  - Cuando `simulcast` se especifica junto con un valor de `maxBitrate` o `maxFramerate`, se espera que la capa de representación más alta se configure teniendo en cuenta estos valores, siempre que `maxBitrate` no sea inferior al valor predeterminado de `maxBitrate` (900 kbps) de la segunda capa más alta del SDK interno.
  - Si `maxBitrate` se especifica como demasiado baja en comparación con el valor predeterminado de la segunda capa más alta, `simulcast` se desactivará.
  - `simulcast` no se puede activar y desactivar sin volver a publicar el contenido multimedia. Para ello, `shouldPublishParticipant` tiene que devolver `false` y llamar a `refreshStrategy`; `shouldPublishParticipant` tiene que devolver `true` y llamar a `refreshStrategy` otra vez.

Obtención de los atributos de los participantes

Si especifica atributos en la solicitud del punto de conexión de `CreateParticipantToken`, puede ver los atributos en las propiedades de `StageParticipantInfo`:

```
stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
   console.log(`Participant ${participant.id} info:`, participant.attributes);
})
```

Gestión de los problemas de red

Cuando se pierde la conexión de red del dispositivo local, el SDK se intenta volver a conectar internamente sin que el usuario lleve a cabo ninguna acción. En algunos casos, el SDK no funciona de manera correcta y es necesario que el usuario actúe.

En términos generales, el estado del escenario se puede gestionar mediante el evento `STAGE_CONNECTION_STATE_CHANGED`:

```
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
   switch (state) {
      case StageConnectionState.DISCONNECTED:
         // handle disconnected UI
         break;
      case StageConnectionState.CONNECTING:
         // handle establishing connection UI
         break;
      case StageConnectionState.CONNECTED:
         // SDK is connected to the Stage
         break;
      case StageConnectionState.ERRORED:
         // unrecoverable error detected, please re-instantiate
         Break;
})
```

En general, el que aparezcan errores después de incorporarse a un escenario correctamente indica que el SDK perdió la conexión y no la pudo restablecer. Cree un nuevo objeto `Stage` e intente incorporarse cuando mejoren las condiciones de la red.

Transmisión del escenario a un canal de IVS

Para transmitir un escenario, cree una sesión de `IVSBroadcastClient` independiente y, a continuación, siga las instrucciones habituales para la transmisión con el SDK, descritas con anterioridad. La lista de `StageStream` expuestas mediante `STAGE_PARTICIPANT_STREAMS_ADDED` se puede utilizar para recuperar las transmisiones multimedia participantes aplicables a la composición del flujo de transmisión, de la siguiente manera:

```
// Setup client with preferred settings
const broadcastClient = getIvsBroadcastClient();

stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {
    streams.forEach(stream => {
        const inputStream = new MediaStream([stream.mediaStreamTrack]);
        switch (stream.streamType) {
            case StreamType.VIDEO:
                broadcastClient.addVideoInputDevice(inputStream, `video-
${participant.id}`, {
```

```
                    index: DESIRED_LAYER,
                    width: MAX_WIDTH,
                    height: MAX_HEIGHT
                });
                break;
            case StreamType.AUDIO:
                broadcastClient.addAudioInputDevice(inputStream, `audio-
${participant.id}`);
                break;
        }
    })
})
```

Si lo desea, puede componer un escenario y transmitirlo a un canal de IVS de baja latencia para llegar a un público más amplio. Consulte Habilitación de varios hosts en una transmisión de Amazon IVS en la Guía del usuario de transmisión de baja latencia.

Problemas conocidos y soluciones alternativas

- Al cerrar las pestañas del navegador o salir de los navegadores sin llamar a `stage.leave()`, los usuarios pueden seguir apareciendo en la sesión con una pantalla congelada o negra durante un máximo de 10 segundos.

  Solución alternativa: ninguna.

- Las sesiones de Safari aparecen de forma intermitente con una pantalla negra para que los usuarios se unan una vez iniciada la sesión.

  Solución alternativa: actualice el navegador y vuelva a conectar la sesión.

- Safari no se recupera correctamente al cambiar de red.

  Solución alternativa: actualice el navegador y vuelva a conectar la sesión.

- La consola para desarrolladores repite un error `Error: UnintentionalError at StageSocket.onClose`.

  Solución alternativa: solo se puede crear un escenario por token de participante. Este error se produce cuando se crea más de una instancia `Stage` con el mismo token de participante, independientemente de si la instancia está en un dispositivo o en varios.

- Es posible que tenga problemas para mantener un `StageParticipantPublishState.PUBLISHED` estado y que reciba

`StageParticipantPublishState.ATTEMPTING_PUBLISH` estados repetidos al escuchar el `StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` evento.

Solución alternativa: limite la resolución de vídeo a 720p al invocar o. `getUserMedia` `getDisplayMedia` En concreto, sus valores de ancho `getUserMedia` y alto y los de `getDisplayMedia` restricción no deben superar los 921600 (1280*720) si se multiplican entre sí.

Limitaciones de Safari

- Para denegar un mensaje de permisos, debe restablecer el permiso en la configuración del sitio web de Safari en el sistema operativo.
- Safari no detecta todos los dispositivos de forma directa con la misma eficacia que Firefox o Chrome. Por ejemplo, no identifica la cámara virtual OBS.

Limitaciones de Firefox

- Los permisos del sistema deben estar habilitados para que Firefox pueda compartir la pantalla. Después de habilitarlos, el usuario debe reiniciar Firefox para que funcione correctamente; de lo contrario, si los permisos se perciben como bloqueados, el navegador emitirá una excepción. NotFoundError
- Falta el método `getCapabilities`. Esto significa que los usuarios no pueden obtener la resolución o la relación de aspecto de la pista multimedia. Consulte este hilo de Bugzilla.
- Faltan varias propiedades `AudioContext`, por ejemplo, la latencia y el recuento de canales. Esto podría suponer un problema para los usuarios avanzados que desean manejar las pistas de audio.

- Las imágenes de la cámara de `getUserMedia` están limitadas a una relación de aspecto 4:3 en MacOS. Consulte el hilo 1 de Bugzilla y el hilo 2 de Bugzilla.

- Con `getDisplayMedia`, la captura de audio no es compatible. Consulte este hilo de Bugzilla.
- La velocidad de fotogramas en la captura de pantalla es poco óptima (¿aproximadamente 15 fps?). Consulte este hilo de Bugzilla.

Limitaciones de la web móvil

- getDisplayMedialos dispositivos móviles no admiten el uso compartido de pantalla.

Solución alternativa: ninguna.

- El participante tarda entre 15 y 30 segundos en salir cuando cierra un navegador sin llamar a `leave()`.

  Solución alternativa: añada una interfaz de usuario que anime a los usuarios a desconectarse correctamente.

- Poner en segundo plano la aplicación hace que se detenga la publicación del vídeo.

  Solución alternativa: muestre una lista de interfaz de usuario cuando el publicador esté en pausa.

- La velocidad de fotogramas del vídeo se reduce durante aproximadamente 5 segundos después de desactivar el silenciamiento de una cámara en los dispositivos Android.

  Solución alternativa: ninguna.

- La transmisión de vídeo se estira al girar para iOS 16.0.

  Solución alternativa: muestre una interfaz de usuario en la que se describa este problema conocido del sistema operativo.

- Al cambiar el dispositivo de entrada de audio, se cambia automáticamente el dispositivo de salida de audio.

  Solución alternativa: ninguna.

- Al poner en segundo plano el navegador, el flujo de publicación se pone en negro y solo produce audio.

  Solución alternativa: ninguna. Esto se debe a motivos de seguridad.

Control de errores

En esta sección se ofrece información general sobre las condiciones de error, cómo el SDK de transmisión web las informa a la aplicación y qué debe hacer una aplicación cuando se producen esos errores. Existen cuatro categorías de errores:

```
try {
  stage = new Stage(token, strategy);
} catch (e) {
  // 1) stage instantiation errors
}
```

```
try {
  await stage.join();
} catch (e) {
  // 2) stage join errors
}

stage.on(StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED, (participantInfo, state)
 => {
  if (state === StageParticipantPublishState.ERRORED) {
    // 3) stage publish errors
  }
});

stage.on(StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED, (participantInfo,
 state) => {
  if (state === StageParticipantSubscribeState.ERRORED) {
    // 4) stage subscribe errors
  }
});
```

Errores de instanciación de etapas

La instanciación por etapas no valida los tokens de forma remota, pero sí comprueba algunos problemas básicos que pueden validarse en el lado del cliente. Como resultado, el SDK puede generar un error.

Token de participante con formato incorrecto

Esto ocurre cuando el token de la etapa tiene un formato incorrecto. Al crear una instancia de una etapa, el SDK muestra un error con este mensaje: “Error al analizar el token de la etapa”..

Acción: cree un token válido e intente la instanciación de nuevo.

Errores al unirse a una etapa

Estos son los errores que se pueden producir al intentar unirse a una etapa por primera vez.

Se eliminó la etapa

Esto ocurre al unirse a una etapa (asociada a un token) que se ha eliminado. El método `join` SDK arroja un error con este mensaje: "InitialConnectTimedOut después de 10 segundos».

Acción: cree un token válido con una nueva etapa e intente unirse de nuevo.

Token de participante caducado

Esto ocurre cuando el token ha caducado. El método de SDK `join` devuelve un error con este mensaje: “El token ha caducado y ya no es válido”.

Acción: cree un token nuevo e intente unirse de nuevo.

El token de participante no es válido o ha sido revocado

Esto ocurre cuando el token no es válido o se revocó o desconectó. El método `join` SDK arroja un error con este mensaje: "InitialConnectTimedOut después de 10 segundos».

Acción: cree un token nuevo e intente unirse de nuevo.

Token desconectado

Esto ocurre cuando el token de la etapa no tiene un formato incorrecto, sino que el servidor de etapas lo rechaza. El método `join` SDK arroja un error con este mensaje: "InitialConnectTimedOut después de 10 segundos».

Acción: cree un token válido e intente unirse de nuevo.

Errores de red durante la unión inicial

Esto ocurre cuando el SDK no puede ponerse en contacto con el servidor de etapas para establecer una conexión. El método `join` SDK arroja un error con este mensaje: "InitialConnectTimedOut después de 10 segundos».

Acción: espere a que se recupere la conectividad del dispositivo e intente conectarse de nuevo.

Errores de red cuando ya está conectado

Si se interrumpe la conexión de red del dispositivo, es posible que el SDK pierda la conexión con los servidores de etapas. Es posible que observe errores en la consola porque el SDK ya no puede acceder a los servicios de backend. Las publicaciones en https://broadcast.stats.live-video.net fallarán.

Si está publicando o suscribiéndose, observará errores en la consola relacionados con los intentos de publicar/suscribirte.

Internamente, el SDK intentará reconectarse con una estrategia de retroceso exponencial.

Acción: espere a que se recupere la conectividad del dispositivo. Si publica o se suscribe, actualice la estrategia para garantizar la republicación de sus transmisiones multimedia.

Errores de publicación y suscripción

Error de publicación: estados de publicación

El SDK informa `ERRORED` cuando se produce un error en una publicación. Esto puede ocurrir debido a las condiciones de la red o si una etapa está llena para los editores.

```
stage.on(StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED, (participantInfo, state)
 => {
  if (state === StageParticipantPublishState.ERRORED) {
      // Handle
  }
});
```

Acción: actualice la estrategia para intentar volver a publicar sus transmisiones multimedia.

Errores de suscriptor

El SDK informa `ERRORED` cuando se produce un error en la suscripción. Esto puede ocurrir debido a las condiciones de la red o si una etapa está llena para los suscriptores.

```
stage.on(StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED, (participantInfo,
 state) => {
  if (state === StageParticipantSubscribeState.ERRORED) {
    // 4) stage subscribe errors
  }
});
```

Acción: actualiza la estrategia para probar una nueva suscripción.

SDK de transmisión de IVS: guía para Android (transmisión en tiempo real)

El SDK de transmisión para transmisión en tiempo real de IVS para Android permite a los participantes enviar y recibir videos en Android.

El paquete de `com.amazonaws.ivs.broadcast` implementa la interfaz descrita en este documento. El SDK admite las siguientes operaciones:

- Incorporación a un escenario
- Publicación de contenido multimedia para otros participantes del escenario
- Suscripción al contenido multimedia de otros participantes del escenario
- Administración y monitoreo del video y audio publicados en el escenario
- Obtención de estadísticas de WebRTC de cada conexión de pares
- Todas las operaciones del SDK de transmisión para Android de transmisión de baja latencia de IVS

Última versión del SDK de transmisión de Android: 1.14.1 (notas de la versión)

Documentación de referencia: para obtener información sobre los métodos más importantes disponibles en el SDK de transmisión para Android de Amazon IVS, consulte la documentación de referencia en https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/android/.

Código de ejemplo: consulte el repositorio de ejemplos de Android en: https://github.com/aws-samples/ -sample. GitHub amazon-ivs-broadcast-android

Requisitos de la plataforma: Android 9.0 y versiones posteriores.

Introducción

Instalación de la biblioteca

A fin de agregar la biblioteca de transmisión de Android de Amazon IVS a su entorno de desarrollo de Android, agregue la biblioteca al archivo `build.gradle` del módulo, como se muestra a continuación (para la versión más reciente del SDK de transmisión de Amazon IVS):

```
repositories {
    mavenCentral()
}

dependencies {
     implementation 'com.amazonaws:ivs-broadcast:1.14.1:stages@aar'
}
```

Agregue el siguiente permiso al manifiesto para permitir que el SDK habilite y deshabilite el altavoz:

```
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
```

Alternativamente, para instalar el SDK de forma manual, descargue la última versión desde esta ubicación:

https://search.maven.org/artifact/com.amazonaws/ivs-broadcast

Asegúrese de descargar el archivo `aar` con `-stages` adjunto.

Solicitar permisos

La aplicación debe solicitar permiso para acceder a la cámara y al micrófono del usuario. (Esto no es específico de Amazon IVS; es necesario para cualquier aplicación que necesite acceso a cámaras y micrófonos).

Aquí, verificamos si el usuario ya ha concedido permisos y, de no ser así, preguntamos por ellos:

```
final String[] requiredPermissions =
         { Manifest.permission.CAMERA, Manifest.permission.RECORD_AUDIO };

for (String permission : requiredPermissions) {
    if (ContextCompat.checkSelfPermission(this, permission)
                != PackageManager.PERMISSION_GRANTED) {
        // If any permissions are missing we want to just request them all.
        ActivityCompat.requestPermissions(this, requiredPermissions, 0x100);
        break;
    }
}
```

Aquí, obtenemos la respuesta del usuario:

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode,
               permissions, grantResults);
    if (requestCode == 0x100) {
        for (int result : grantResults) {
            if (result == PackageManager.PERMISSION_DENIED) {
```

```
                return;
            }
        }
        setupBroadcastSession();
    }
}
```

Publicación y suscripción

Conceptos

Los siguientes tres conceptos básicos subyacen a la funcionalidad de transmisión en tiempo real: escenario, estrategia y renderizador. El objetivo del diseño es minimizar la cantidad de lógica necesaria por parte del cliente para crear un producto que funcione.

Escenario

La clase `Stage` es el principal punto de interacción entre la aplicación host y el SDK. Representa el escenario como tal y se usa para entrar y salir de él. Para crear un escenario e incorporarse a él, es necesaria una cadena de símbolos válida y que no haya vencido del plano de control (representada como `token`). Entrar a un escenario y salir de él es sencillo.

```
Stage stage = new Stage(context, token, strategy);

try {
 stage.join();
} catch (BroadcastException exception) {
 // handle join exception
}

stage.leave();
```

También se puede adjuntar `StageRenderer` en la clase `Stage`:

```
stage.addRenderer(renderer); // multiple renderers can be added
```

Strategy (Estrategia)

La interfaz `Stage.Strategy` proporciona una forma para que la aplicación host comunique el estado deseado del escenario al SDK. Es necesario implementar las siguientes tres

funciones: `shouldSubscribeToParticipant`, `shouldPublishFromParticipant` y `stageStreamsToPublishForParticipant`. Todas se analizan a continuación.

Suscripción a participantes

```
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo);
```

Cuando un participante remoto se incorpora al escenario, el SDK consulta la aplicación host sobre el estado de suscripción deseado de ese participante. Las opciones son `NONE`, `AUDIO_ONLY` y `AUDIO_VIDEO`. Al devolver un valor para esta función, la aplicación host no tiene que preocuparse por el estado de la publicación, el estado actual de la suscripción ni el estado de la conexión del escenario. Si se devuelve `AUDIO_VIDEO`, el SDK espera a que el participante remoto publique antes de suscribirse y actualiza la aplicación host a través del renderizador durante todo el proceso.

Este es un ejemplo de implementación:

```
@Override
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo) {
 return Stage.SubscribeType.AUDIO_VIDEO;
}
```

Esta es la implementación completa de esta función para una aplicación host que siempre quiere que todos los participantes se vean entre sí; por ejemplo, una aplicación de videochat.

También son posibles implementaciones más avanzadas. Utilice la propiedad `userInfo` en `ParticipantInfo` para suscribirse de forma selectiva a los participantes en función de los atributos proporcionados por el servidor:

```
@Override
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo) {
 switch(participantInfo.userInfo.get(“role”)) {
  case “moderator”:
   return Stage.SubscribeType.NONE;
  case “guest”:
   return Stage.SubscribeType.AUDIO_VIDEO;
  default:
   return Stage.SubscribeType.NONE;
 }
```

```
}
```

Esto se puede utilizar para crear un escenario en el que los moderadores puedan monitorear a todos los invitados sin que ellos mismos los vean ni los escuchen. La aplicación host podría utilizar una lógica empresarial adicional para permitir que los moderados se vean entre sí, pero permanezcan invisibles para los invitados.

Publicación

```
boolean shouldPublishFromParticipant(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo);
```

Una vez realizada la conexión al escenario, el SDK consulta la aplicación host para ver si un participante en particular tiene que publicar. Esto solo se invoca en los participantes locales que tienen permiso para publicar en función del token proporcionado.

Este es un ejemplo de implementación:

```
@Override
boolean shouldPublishFromParticipant(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo) {
 return true;
}
```

Esto es para una aplicación de videochat estándar en la que los usuarios siempre quieren publicar. Pueden activar y desactivar el sonido y el video para ocultarse o verse y escucharse al instante. (También pueden usar la opción de publicar o anular la publicación, pero esto es mucho más lento. Es preferible silenciar o activar el sonido en casos de uso en los que se quiera cambiar la visibilidad de manera frecuente).

Elección de las transmisiones que publicar

```
@Override
List<LocalStageStream> stageStreamsToPublishForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo);
}
```

Al publicar, esto se utiliza para determinar qué transmisiones de audio y video se tienen que publicar. Esto se explica en mayor detalle más adelante en Publicación de una transmisión multimedia.

Actualización de la estrategia

La estrategia pretende ser dinámica: los valores devueltos por cualquiera de las funciones anteriores se pueden cambiar en cualquier momento. Por ejemplo, si la aplicación host no quiere publicar hasta que el usuario final presione un botón, puede devolver una variable de `shouldPublishFromParticipant` (como `hasUserTappedPublishButton`). Cuando esa variable cambie en función de una interacción del usuario final, llame a `stage.refreshStrategy()` para indicar al SDK que debe consultar la estrategia a fin de obtener los valores más recientes y aplicar solo los cambios. Si el SDK observa que el valor `shouldPublishFromParticipant` cambió, iniciará el proceso de publicación. Si las consultas del SDK y todas las funciones devuelven el mismo valor que antes, la llamada a `refreshStrategy` no hará ninguna modificación en el escenario.

Si el valor devuelto de `shouldSubscribeToParticipant` cambia de `AUDIO_VIDEO` a `AUDIO_ONLY`, la transmisión de video se eliminará para todos los participantes con valores devueltos modificados, si ya existía con anterioridad una transmisión de video.

Por lo general, el escenario utiliza la estrategia para aplicar de la manera más eficiente la diferencia entre las estrategias anteriores y actuales, sin que la aplicación host tenga que preocuparse por todo el estado necesario para administrarla correctamente. Por eso, piense en la llamada a `stage.refreshStrategy()` como una operación barata, porque no hace nada a no ser que cambie la estrategia.

Renderer

La interfaz `StageRenderer` comunica el estado del escenario a la aplicación host. Por lo general, los eventos proporcionados por el renderizador permiten el funcionamiento completo de las actualizaciones de la interfaz de usuario de la aplicación host. El renderizador ofrece el siguiente resultado:

```
void onParticipantJoined(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo);

void onParticipantLeft(@NonNull Stage stage, @NonNull ParticipantInfo participantInfo);

void onParticipantPublishStateChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull Stage.PublishState publishState);

void onParticipantSubscribeStateChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull Stage.SubscribeState subscribeState);
```

```
void onStreamsAdded(@NonNull Stage stage, @NonNull ParticipantInfo participantInfo,
 @NonNull List<StageStream> streams);

void onStreamsRemoved(@NonNull Stage stage, @NonNull ParticipantInfo participantInfo,
 @NonNull List<StageStream> streams);

void onStreamsMutedChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull List<StageStream> streams);

void onError(@NonNull BroadcastException exception);

void onConnectionStateChanged(@NonNull Stage stage, @NonNull Stage.ConnectionState
 state, @Nullable BroadcastException exception);
```

Para la mayoría de estos métodos, se proporcionan el `Stage` y la `ParticipantInfo` correspondientes.

No se espera que la información que proporciona el renderizador afecte a los valores de retorno de la estrategia. Por ejemplo, no se espera que el valor devuelto de `shouldSubscribeToParticipant` cambie cuando se llama a `onParticipantPublishStateChanged`. Si la aplicación host quiere suscribirse a un participante en particular, debe devolver el tipo de suscripción deseado, independientemente del estado de publicación de ese participante. El SDK es responsable de garantizar que se aplique el estado deseado de la estrategia en el momento correcto según el estado del escenario.

`StageRenderer` se puede adjuntar a la clase del escenario:

```
stage.addRenderer(renderer); // multiple renderers can be added
```

Tenga en cuenta que solo los participantes que publican desencadenan `onParticipantJoined`. `onParticipantLeft` se desencadena cada vez que un participante deja de publicar o abandona la sesión del escenario.

Publicación de una transmisión multimedia

Los dispositivos locales, como las cámaras y los micrófonos integrados, se detectan a través de `DeviceDiscovery`. A continuación, se muestra un ejemplo de cómo seleccionar la cámara frontal y el micrófono predeterminados y devolverlos como `LocalStageStreams` para que los publique el SDK:

```
DeviceDiscovery deviceDiscovery = new DeviceDiscovery(context);
```

```
List<Device> devices = deviceDiscovery.listLocalDevices();
List<LocalStageStream> publishStreams = new ArrayList<LocalStageStream>();

Device frontCamera = null;
Device microphone = null;

// Create streams using the front camera, first microphone
for (Device device : devices) {
 Device.Descriptor descriptor = device.getDescriptor();
 if (!frontCamera && descriptor.type == Device.Descriptor.DeviceType.Camera &&
 descriptor.position = Device.Descriptor.Position.FRONT) {
  front Camera = device;
 }
 if (!microphone && descriptor.type == Device.Descriptor.DeviceType.Microphone) {
  microphone = device;
 }
}

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera);
AudioLocalStageStream microphoneStream = new AudioLocalStageStream(microphoneDevice);

publishStreams.add(cameraStream);
publishStreams.add(microphoneStream);

// Provide the streams in Stage.Strategy
@Override
@NonNull List<LocalStageStream> stageStreamsToPublishForParticipant(@NonNull Stage
 stage, @NonNull ParticipantInfo participantInfo) {
 return publishStreams;
}
```

Visualización y eliminación de participantes

Cuando se complete la suscripción, recibirá una matriz de objetos `StageStream` a través de la función on`StreamsAdded` del renderizador. Puede obtener la vista previa de una `ImageStageStream`:

```
ImagePreviewView preview = ((ImageStageStream)stream).getPreview();

// Add the view to your view hierarchy
LinearLayout previewHolder = findViewById(R.id.previewHolder);
preview.setLayoutParams(new LinearLayout.LayoutParams(
```

```
  LinearLayout.LayoutParams.MATCH_PARENT,
  LinearLayout.LayoutParams.MATCH_PARENT));
previewHolder.addView(preview);
```

Puede recuperar las estadísticas de audio de una `AudioStageStream`:

```
((AudioStageStream)stream).setStatsCallback((peak, rms) -> {
 // handle statistics
});
```

Cuando un participante deja de publicar o anula su suscripción, se invoca la función `onStreamsRemoved` con las transmisiones que se eliminaron. Las aplicaciones host tienen que usar esto como una señal para eliminar la transmisión de video del participante de la jerarquía de visualización.

`onStreamsRemoved` se invoca para todas las situaciones en las que se puede eliminar una transmisión, como las siguientes:

- El participante remoto deja de publicar.
- Un dispositivo local cancela la suscripción o cambia la suscripción de `AUDIO_VIDEO` a `AUDIO_ONLY`.
- El participante remoto abandona el escenario.
- El participante local abandona el escenario.

Ya que `onStreamsRemoved` se invoca en todas las situaciones, no es necesaria una lógica empresarial personalizada para eliminar a los participantes de la interfaz de usuario durante las operaciones de licencia remota o local.

Activación y desactivación del sonido de las transmisiones multimedia

Los objetos de `LocalStageStream` tienen una función `setMuted` que controla si la transmisión está silenciada. Esta función se puede invocar en la transmisión antes o después de que la devuelva la función de estrategia `streamsToPublishForParticipant`.

Importante: Si `streamsToPublishForParticipant` devuelve una nueva instancia de objeto de `LocalStageStream` después de una llamada a `refreshStrategy`, el estado de sonido desactivado del nuevo objeto de transmisión se aplicará al escenario. Tenga cuidado al crear nuevas instancias `LocalStageStream` para asegurarse de que se mantenga el estado de sonido desactivado que se espera.

Monitoreo del estado de sonido desactivado en los contenidos multimedia del participante remoto

Cuando un participante cambia el estado de sonido desactivado de su transmisión de video o audio, se invoca la función onStreamMutedChanged de renderizado con una lista de las transmisiones que cambiaron. Use el método getMuted en StageStream para actualizar la interfaz de usuario según corresponda.

```
@Override
void onStreamsMutedChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull List<StageStream> streams) {
 for (StageStream stream : streams) {
  boolean muted = stream.getMuted();
  // handle UI changes
 }
}
```

Obtención de estadísticas de WebRTC

Para obtener las estadísticas más recientes de WebRTC de una transmisión de publicación o de suscripción, utilice requestRTCStats en StageStream. Cuando se complete una recopilación, recibirá estadísticas a través de StageStream.Listener, que se puede configurar en StageStream.

```
stream.requestRTCStats();

@Override
void onRTCStats(Map<String, Map<String, String>> statsMap) {
 for (Map.Entry<String, Map<String, string>> stat : statsMap.entrySet()) {
  for(Map.Entry<String, String> member : stat.getValue().entrySet()) {
   Log.i(TAG, stat.getKey() + " has member " + member.getKey() + " with value " +
 member.getValue());
  }
 }
}
```

Obtención de los atributos de los participantes

Si especifica atributos en la solicitud del punto de conexión de CreateParticipantToken, puede ver los atributos en las propiedades de ParticipantInfo:

```
@Override
void onParticipantJoined(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo) {
 for (Map.Entry<String, String> entry : participantInfo.userInfo.entrySet()) {
  Log.i(TAG, “attribute: “ + entry.getKey() + “ = “ + entry.getValue());
 }
}
```

Continuación de la sesión en segundo plano

Cuando la aplicación pase a segundo plano, es posible que quiera dejar de publicar o suscribirse solo al audio de otros participantes remotos. Para ello, actualice la implementación `Strategy` para detener la publicación y suscríbase a `AUDIO_ONLY` (o a `NONE`, si corresponde).

```
// Local variables before going into the background
boolean shouldPublish = true;
Stage.SubscribeType subscribeType = Stage.SubscribeType.AUDIO_VIDEO;

// Stage.Strategy implementation
@Override
boolean shouldPublishFromParticipant(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo) {
 return shouldPublish;
}

@Override
Stage.SubscribeType shouldSubscribeToParticipant(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo) {
 return subscribeType;
}

// In our Activity, modify desired publish/subscribe when we go to background, then
 call refreshStrategy to update the stage
@Override
void onStop() {
 super.onStop();
 shouldPublish = false;
 subscribeTpye = Stage.SubscribeType.AUDIO_ONLY;
 stage.refreshStrategy();
}
```

Activación o desactivación de la codificación en capas con transmisión simultánea

Al publicar una transmisión multimedia, el SDK transmite transmisión de video de alta y baja calidad, de modo que los participantes remotos puedan suscribirse a transmisión incluso si tienen un ancho de banda de enlace descendente limitado. La codificación en capas con transmisión simultánea está activada de forma predeterminada. Puede desactivarla mediante la clase `StageVideoConfiguration.Simulcast`:

```
// Disable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(false);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

Limitaciones de la configuración de video

El SDK no permite forzar el uso del modo vertical ni del horizontal mediante `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)`. En la orientación vertical, la dimensión más pequeña se utiliza como el ancho; en la orientación horizontal, como la altura. Esto significa que las dos siguientes llamadas a `setSize` tendrán el mismo efecto en la configuración de video:

```
StageVideo Configuration config = new StageVideo Configuration();

config.setSize(BroadcastConfiguration.Vec2(720f, 1280f);
config.setSize(BroadcastConfiguration.Vec2(1280f, 720f);
```

Gestión de los problemas de red

Cuando se pierde la conexión de red del dispositivo local, el SDK se intenta volver a conectar internamente sin que el usuario lleve a cabo ninguna acción. En algunos casos, el SDK no funciona de manera correcta y es necesario que el usuario actúe. Hay dos errores principales relacionados con la pérdida de la conexión de red:

- Código de error 1400, mensaje: «PeerConnection se ha perdido debido a un error de red desconocido»
- Código de error 1300, mensaje: “Se han agotado los reintentos”

Si se recibe el primer error, pero no el segundo, el SDK sigue conectado al escenario e intentará restablecer sus conexiones automáticamente. Como medida de seguridad, puede hacer una llamada a `refreshStrategy` sin cambiar los valores devueltos por el método de estrategia para iniciar un intento de reconexión manual.

Si se recibe el segundo error, los intentos de reconexión del SDK fallaron y el dispositivo local ya no está conectado al escenario. En este caso, intente reincorporarse al escenario con una llamada a `join` después de que se restablezca la conexión de red.

En general, el que aparezcan errores después de incorporarse a un escenario correctamente indica que el SDK no pudo restablecer la conexión. Cree un nuevo objeto `Stage` e intente incorporarse cuando mejoren las condiciones de la red.

Uso de micrófonos Bluetooth

Para publicar con dispositivos de micrófono Bluetooth, debe iniciar una conexión SCO Bluetooth:

```
Bluetooth.startBluetoothSco(context);
// Now bluetooth microphones can be used
…
// Must also stop bluetooth SCO
Bluetooth.stopBluetoothSco(context);
```

Problemas conocidos y soluciones alternativas

- Cuando un dispositivo Android se pone en reposo y se activa, es posible que la vista previa esté congelada.

  Solución alternativa: cree y utilice un nuevo `Stage`.
- Cuando un participante se incorpora con un token que utiliza otro participante, la primera conexión se pierde sin que se produzca un error específico.

  Solución alternativa: ninguna.
- Hay un problema poco frecuente en el que el editor publica, pero el estado de publicación que reciben los suscriptores es `inactive`.

  Solución alternativa: pruebe a salir de la sesión y, a continuación, reincorporarse. Si el problema persiste, cree un nuevo token para el publicador.

- Durante una sesión del escenario, se puede producir un problema intermitente y poco frecuente de distorsión de audio, por lo general en llamadas de mayor duración.

  Solución alternativa: el participante con audio distorsionado puede abandonar la sesión y volver a unirse a ella o anular la publicación y volver a publicar su audio para solucionar el problema.
- No se admiten micrófonos externos cuando se hacen publicaciones en un escenario.

  Solución alternativa: no utilice un micrófono externo conectado por USB para publicar en un escenario.
- No se admite el uso de `createSystemCaptureSources` para publicar en un escenario con pantalla compartida.

  Solución alternativa: administre la captura del sistema de forma manual mediante orígenes de entrada de imágenes personalizados y orígenes de entrada de audio personalizados.
- Cuando se elimina una `ImagePreviewView` de un elemento principal (por ejemplo, se llama a `removeView()` en el elemento principal), `ImagePreviewView` se libera inmediatamente. La `ImagePreviewView` no muestra ningún fotograma cuando se agrega a otra vista principal.

  Solución alternativa: solicite otra vista previa mediante `getPreview`.
- Al incorporarse a un escenario con un Samsung Galaxy S22/+ con Android 12, es posible que aparezca un error 1401 y que el dispositivo local no pueda incorporarse al escenario o lo haga, pero no haya audio.

  Solución alternativa: actualice a Android 13.
- Al incorporarse a un escenario con un Nokia X20 en Android 13, es posible que la cámara no se abra y se produzca una excepción.

  Solución alternativa: ninguna.
- Es posible que los dispositivos con el chipset MediaTek Helio no reproduzcan correctamente el vídeo de los participantes remotos.

  Solución alternativa: ninguna.
- En algunos dispositivos, el sistema operativo del dispositivo puede elegir un micrófono diferente al seleccionado a través del SDK. Esto se debe a que el SDK de transmisión de Amazon IVS no puede controlar la forma en que se define la ruta de audio de `VOICE_COMMUNICATION`, ya que varía según los distintos fabricantes de dispositivos.

  Solución alternativa: ninguna.

- Algunos codificadores de vídeo Android no se pueden configurar con un tamaño de vídeo inferior a 176 x 176. La configuración de un tamaño más pequeño provoca un error e impide la transmisión.

  Solución alternativa: no configure el tamaño del vídeo para que sea inferior a 176 x 176.

Control de errores

Errores fatales frente a errores no fatales

El objeto de error tiene un campo booleano “es grave” de `BroadcastException`.

En general, los errores fatales están relacionados con la conexión al servidor de etapas (o bien no se puede establecer una conexión o bien se pierde y no se puede recuperar). La aplicación debe volver a crear el escenario y volver a unirse, posiblemente con un nuevo token o cuando se recupere la conectividad del dispositivo.

Los errores no fatales suelen estar relacionados con el estado de publicación/suscripción y son gestionados por el SDK, que reintenta la operación de publicación/suscripción.

Puede comprobar esta propiedad:

```
try {
  stage.join(...)
} catch (e: BroadcastException) {
  If (e.isFatal) {
    // the error is fatal
```

Errores de unión

Token con formato incorrecto

Esto ocurre cuando el token de la etapa tiene un formato incorrecto.

El SDK genera una excepción de Java al llamar a `stage.join`, con el código de error =1000 y fatal =true.

Acción: cree un token válido e intente unirse de nuevo.

Token vencido

Esto ocurre cuando el token de la etapa está caducado.

El SDK genera una excepción de Java al llamar a `stage.join`, con el código de error =1001 y fatal =true.

Acción: cree un token nuevo e intente unirse de nuevo.

Token no válido o revocado

Esto ocurre cuando el token de la etapa no tiene un formato incorrecto, sino que el servidor de etapas lo rechaza. Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK llama a `onConnectionStateChanged` con una excepción, con el código de error =1026 y fatal =true.

Acción: cree un token válido e intente unirse de nuevo.

Errores de red durante la unión inicial

Esto ocurre cuando el SDK no puede ponerse en contacto con el servidor de etapas para establecer una conexión. Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK llama a `onConnectionStateChanged` con una excepción, con el código de error =1300 y fatal =true.

Acción: espere a que se recupere la conectividad del dispositivo e intente conectarse de nuevo.

Errores de red cuando ya está conectado

Si se interrumpe la conexión de red del dispositivo, es posible que el SDK pierda la conexión con los servidores de etapas. Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK llama a `onConnectionStateChanged` con una excepción, con el código de error =1300 y fatal =true.

Acción: espere a que se recupere la conectividad del dispositivo e intente conectarse de nuevo.

Errores de publicación/suscripción

Inicial

Existen varios errores:

- MultihostSessionOfferCreationFailPublish (1020)
- MultihostSessionOfferCreationFailSubscribe (1021)
- MultihostSessionNoIceCandidates (1022)
- MultihostSessionStageAtCapacity (1024)
- SignallingSessionCannotRead (1201)
- SignallingSessionCannotSend (1202)
- SignallingSessionBadResponse (1203)

Estos se informan de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK vuelve a intentar la operación un número limitado de veces. Durante los reintentos, el estado de publicación/suscripción es `ATTEMPTING_PUBLISH`/`ATTEMPTING_SUBSCRIBE`. Si el reintento se realiza correctamente, el estado cambia a `PUBLISHED`/`SUBSCRIBED`.

El SDK llama a `onError` con el código de error correspondiente y fatal =false.

Acción: no es necesario realizar ninguna acción, ya que el SDK vuelve a intentarlo automáticamente. Si lo desea, la aplicación puede actualizar la estrategia para forzar más reintentos.

Ya está establecido, luego falla

Una publicación o una suscripción pueden fallar una vez establecidas, muy probablemente debido a un error de red. El código de error para “Se ha perdido la conexión de pares debido a un error de red” es 1400.

Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK vuelve a intentar la operación de publicación/suscripción. Durante los reintentos, el estado de publicación/suscripción es `ATTEMPTING_PUBLISH`/`ATTEMPTING_SUBSCRIBE`. Si el reintento se realiza correctamente, el estado cambia a `PUBLISHED`/`SUBSCRIBED`.

El SDK llama a `onError` con el código de error =1400 y fatal =false.

Acción: no es necesario realizar ninguna acción, ya que el SDK vuelve a intentarlo automáticamente. Si lo desea, la aplicación puede actualizar la estrategia para forzar más reintentos. En caso de pérdida total de conectividad, es probable que la conexión a las etapas también falle.

SDK de transmisión de IVS: guía para iOS (transmisión en tiempo real)

El SDK de transmisión para transmisión en tiempo real de IVS para iOS permite a los participantes enviar y recibir videos en iOS.

El módulo `AmazonIVSBroadcast` implementa la interfaz descrita en este documento. Se admiten las siguientes operaciones:

- Incorporación a un escenario
- Publicación de contenido multimedia para otros participantes del escenario
- Suscripción al contenido multimedia de otros participantes del escenario
- Administración y monitoreo del video y audio publicados en el escenario
- Obtención de estadísticas de WebRTC de cada conexión de pares
- Todas las operaciones del SDK de transmisión para iOS de transmisión de baja latencia de IVS

Última versión del SDK de transmisión para iOS: 1.14.1 (notas de la versión)

Documentación de referencia: para obtener información sobre los métodos más importantes disponibles en el SDK de transmisión para iOS de Amazon IVS, consulte la documentación de referencia en https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios/.

Código de ejemplo: consulte el repositorio de ejemplos de iOS en GitHub: https://github.com/aws-samples/ amazon-ivs-broadcast-ios -sample.

Requisitos de la plataforma: iOS 14 o posterior

Introducción

Instalación de la biblioteca

Le recomendamos que integre el SDK de transmisión mediante CocoaPods. (También puede agregar el marco a su proyecto de forma manual).

Recomendado: integre el SDK de transmisión (CocoaPods)

La funcionalidad en tiempo real se publica como una subespecificación del SDK de transmisión para transmisión de baja latencia de iOS. De este modo, los clientes pueden elegir incluirla o excluirla en función de sus necesidades. Incluirla aumenta el tamaño del paquete.

Los lanzamientos se publican CocoaPods con el nombre`AmazonIVSBroadcast`. Agregue esta dependencia a su Podfile:

```
pod 'AmazonIVSBroadcast/Stages'
```

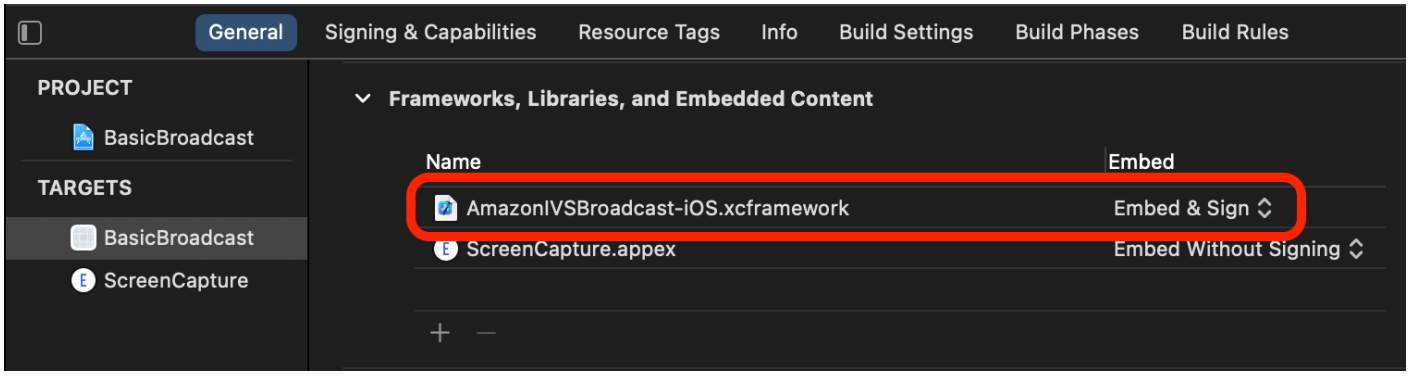
Ejecute `pod install` y el SDK estará disponible en su `.xcworkspace`.

Importante: El SDK de transmisión para transmisión en tiempo real de IVS (es decir, con la subespecificación de escenario) incluye todas las características del SDK de transmisión para transmisión de baja latencia de IVS. No es posible integrar ambos SDK en el mismo proyecto. Si añades la subespecificación de la etapa vía CocoaPods a tu proyecto, asegúrate de eliminar cualquier otra línea del Podfile que la contenga. `AmazonIVSBroadcast` Por ejemplo, no incluya estas dos líneas en el Podfile:

```
pod 'AmazonIVSBroadcast'
pod 'AmazonIVSBroadcast/Stages'
```

Método alternativo: instalar el marco de forma manual

1. Descarga la última versión desde https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Extraiga el contenido del archivo. `AmazonIVSBroadcast.xcframework` contiene el SDK para el dispositivo y el simulador.
3. Integre el `AmazonIVSBroadcast.xcframework` arrastrándolo a la sección Frameworks, Libraries, and Embedded Content (Marcos, bibliotecas y contenido integrado) de la pestaña General para el destino de la aplicación.



Solicitar permisos

La aplicación debe solicitar permiso para acceder a la cámara y al micrófono del usuario. (Esto no es específico de Amazon IVS; es necesario para cualquier aplicación que necesite acceso a cámaras y micrófonos).

Aquí, verificamos si el usuario ya ha concedido permisos y, de no ser así, preguntamos por ellos:

```
switch AVCaptureDevice.authorizationStatus(for: .video) {
case .authorized: // permission already granted.
case .notDetermined:
   AVCaptureDevice.requestAccess(for: .video) { granted in
       // permission granted based on granted bool.
   }
case .denied, .restricted: // permission denied.
@unknown default: // permissions unknown.
}
```

Es necesario hacer esto para los tipos de medios `.video` y `.audio`, si desea tener acceso a cámaras y micrófonos, respectivamente.

También es necesario agregar entradas para `NSCameraUsageDescription` y `NSMicrophoneUsageDescription` para su `Info.plist`. De lo contrario, la aplicación se bloqueará al intentar solicitar permisos.

Desactivar el temporizador inactivo de la aplicación

Esto es opcional, pero recomendable. Evita que el dispositivo se ponga en suspensión mientras utiliza el SDK de transmisión, lo que interrumpiría la transmisión.

```
override func viewDidAppear(_ animated: Bool) {
   super.viewDidAppear(animated)
   UIApplication.shared.isIdleTimerDisabled = true
}
override func viewDidDisappear(_ animated: Bool) {
   super.viewDidDisappear(animated)
   UIApplication.shared.isIdleTimerDisabled = false
}
```

Publicación y suscripción

Conceptos

Los siguientes tres conceptos básicos subyacen a la funcionalidad de transmisión en tiempo real: escenario, estrategia y renderizador. El objetivo del diseño es minimizar la cantidad de lógica necesaria por parte del cliente para crear un producto que funcione.

Escenario

La clase `IVSStage` es el principal punto de interacción entre la aplicación host y el SDK. La clase representa el escenario como tal y se usa para entrar y salir de él. Para crear un escenario o incorporarse a él, es necesaria una cadena de símbolos válida y que no haya vencido del plano de control (representada como `token`). Entrar a un escenario y salir de él es sencillo.

```
let stage = try IVSStage(token: token, strategy: self)

try stage.join()

stage.leave()
```

También se pueden adjuntar `IVSStageRenderer` y `IVSErrorDelegate` en la clase `IVSStage`:

```
let stage = try IVSStage(token: token, strategy: self)
stage.errorDelegate = self
stage.addRenderer(self) // multiple renderers can be added
```

Strategy (Estrategia)

El protocolo `IVSStageStrategy` proporciona una forma para que la aplicación host comunique el estado deseado del escenario al SDK. Es necesario implementar las siguientes tres funciones: `shouldSubscribeToParticipant`, `shouldPublishParticipant` y `streamsToPublishForParticipant`. Todas se analizan a continuación.

Suscripción a participantes

```
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType
```

Cuando un participante remoto se une a un escenario, el SDK consulta a la aplicación host sobre el estado de suscripción deseado para ese participante. Las opciones son `.none`, `.audioOnly` y `.audioVideo`. Al devolver un valor para esta función, la aplicación host no tiene que preocuparse por el estado de la publicación, el estado actual de la suscripción ni el estado de la conexión del escenario. Si se devuelve `.audioVideo`, el SDK espera a que el participante remoto publique antes de suscribirse y actualiza la aplicación host a través del renderizador durante todo el proceso.

Este es un ejemplo de implementación:

```
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
    return .audioVideo
}
```

Esta es la implementación completa de esta función para una aplicación host que siempre quiere que todos los participantes se vean entre sí; por ejemplo, una aplicación de videochat.

También son posibles implementaciones más avanzadas. Utilice la propiedad `attributes` en `IVSParticipantInfo` para suscribirse de forma selectiva a los participantes en función de los atributos proporcionados por el servidor:

```
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
    switch participant.attributes["role"] {
    case "moderator": return .none
    case "guest": return .audioVideo
    default: return .none
    }
}
```

Esto se puede utilizar para crear un escenario en el que los moderadores puedan monitorear a todos los invitados sin que ellos mismos los vean ni los escuchen. La aplicación host podría utilizar una lógica empresarial adicional para permitir que los moderadores se vean entre sí, pero permanezcan invisibles para los invitados.

Publicación

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool
```

Una vez realizada la conexión al escenario, el SDK consulta la aplicación host para ver si un participante en particular tiene que publicar. Esto solo se invoca en los participantes locales que tienen permiso para publicar en función del token proporcionado.

Este es un ejemplo de implementación:

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool {
    return true
}
```

Esto es para una aplicación de videochat estándar en la que los usuarios siempre quieren publicar. Pueden activar y desactivar el sonido y el video para ocultarse o verse y escucharse al instante. (También pueden usar la opción de publicar o anular la publicación, pero esto es mucho más lento. Es preferible silenciar o activar el sonido en casos de uso en los que se quiera cambiar la visibilidad de manera frecuente).

Elección de las transmisiones que publicar

```
func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream]
```

Al publicar, esto se utiliza para determinar qué transmisiones de audio y video se tienen que publicar. Esto se explica en mayor detalle más adelante en Publicación de una transmisión multimedia.

Actualización de la estrategia

La estrategia pretende ser dinámica: los valores devueltos por cualquiera de las funciones anteriores se pueden cambiar en cualquier momento. Por ejemplo, si la aplicación host no quiere publicar hasta que el usuario final presione un botón, puede devolver una variable de `shouldPublishParticipant` (como `hasUserTappedPublishButton`). Cuando esa variable cambie en función de una interacción del usuario final, llame a `stage.refreshStrategy()` para indicar al SDK que debe consultar la estrategia a fin de obtener los valores más recientes y aplicar solo los cambios. Si el SDK observa que el valor `shouldPublishParticipant` cambió, iniciará el proceso de publicación. Si las consultas del SDK y todas las funciones devuelven el mismo valor que antes, la llamada `refreshStrategy` no hará ninguna modificación en el escenario.

Si el valor devuelto de `shouldSubscribeToParticipant` cambia de `.audioVideo` a `.audioOnly`, la transmisión de video se eliminará para todos los participantes con valores devueltos modificados, si ya existía con anterioridad una transmisión de video.

Por lo general, el escenario utiliza la estrategia para aplicar de la manera más eficiente la diferencia entre las estrategias anteriores y actuales, sin que la aplicación host tenga que preocuparse por todo el estado necesario para administrarla correctamente. Por eso, piense en la llamada a `stage.refreshStrategy()` como una operación barata, porque no hace nada a no ser que cambie la estrategia.

Renderer

El protocolo `IVSStageRenderer` comunica el estado del escenario a la aplicación host. Por lo general, los eventos proporcionados por el renderizador permiten el funcionamiento completo de las actualizaciones de la interfaz de usuario de la aplicación host. El renderizador ofrece el siguiente resultado:

```
func stage(_ stage: IVSStage, participantDidJoin participant: IVSParticipantInfo)

func stage(_ stage: IVSStage, participantDidLeave participant: IVSParticipantInfo)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange publishState:
 IVSParticipantPublishState)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChange
 subscribeState: IVSParticipantSubscribeState)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didAdd streams:
 [IVSStageStream])

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didRemove streams:
 [IVSStageStream])

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChangeMutedStreams
 streams: [IVSStageStream])

func stage(_ stage: IVSStage, didChange connectionState: IVSStageConnectionState,
 withError error: Error?)
```

No se espera que la información que proporciona el renderizador afecte a los valores de retorno de la estrategia. Por ejemplo, no se espera que el valor devuelto de `shouldSubscribeToParticipant` cambie cuando se llama a `participant:didChangePublishState`. Si la aplicación host quiere suscribirse a un participante en particular, debe devolver el tipo de suscripción deseado, independientemente del estado de publicación de ese participante. El SDK es responsable de

garantizar que se aplique el estado deseado de la estrategia en el momento correcto según el estado del escenario.

Tenga en cuenta que solo los participantes que publican desencadenan `participantDidJoin`. `participantDidLeave` se desencadena cada vez que un participante deja de publicar o abandona la sesión del escenario.

Publicación de una transmisión multimedia

Los dispositivos locales, como las cámaras y los micrófonos integrados, se detectan a través de `IVSDeviceDiscovery`. A continuación, se muestra un ejemplo de cómo seleccionar la cámara frontal y el micrófono predeterminados y devolverlos como `IVSLocalStageStreams` para que los publique el SDK:

```
let devices = IVSDeviceDiscovery.listLocalDevices()

// Find the camera virtual device, choose the front source, and create a stream
let camera = devices.compactMap({ $0 as? IVSCamera }).first!
let frontSource = camera.listAvailableInputSources().first(where: { $0.position
 == .front })!
camera.setPreferredInputSource(frontSource)
let cameraStream = IVSLocalStageStream(device: camera)

// Find the microphone virtual device, enable echo cancellation, and create a stream
let microphone = devices.compactMap({ $0 as? IVSMicrophone }).first!
microphone.isEchoCancellationEnabled = true
let microphoneStream = IVSLocalStageStream(device: microphone)

// This is a function on IVSStageStrategy
func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream] {
    return [cameraStream, microphoneStream]
}
```

Visualización y eliminación de participantes

Cuando se complete la suscripción, recibirá una matriz de objetos `IVSStageStream` a través de la función `didAddStreams` del renderizador. Para obtener una vista previa o recibir estadísticas del audio de este participante, puede acceder al objeto `IVSDevice` subyacente desde la transmisión:

```
if let imageDevice = stream.device as? IVSImageDevice {
```

```
    let preview = imageDevice.previewView()
    /* attach this UIView subclass to your view */
} else if let audioDevice = stream.device as? IVSAudioDevice {
    audioDevice.setStatsCallback( { stats in
        /* process stats.peak and stats.rms */
    })
}
```

Cuando un participante deja de publicar o anula su suscripción, se invoca la función `didRemoveStreams` con las transmisiones que se eliminaron. Las aplicaciones host tienen que usar esto como una señal para eliminar la transmisión de video del participante de la jerarquía de visualización.

`didRemoveStreams` se invoca para todas las situaciones en las que se puede eliminar una transmisión, como las siguientes:

- El participante remoto deja de publicar.
- Un dispositivo local cancela la suscripción o cambia la suscripción de `.audioVideo` a `.audioOnly`.
- El participante remoto abandona el escenario.
- El participante local abandona el escenario.

Ya que `didRemoveStreams` se invoca en todas las situaciones, no es necesaria una lógica empresarial personalizada para eliminar a los participantes de la interfaz de usuario durante las operaciones de licencia remota o local.

Activación y desactivación del sonido de las transmisiones multimedia

Los objetos de `IVSLocalStageStream` tienen una función `setMuted` que controla si la transmisión está silenciada. Esta función se puede invocar en la transmisión antes o después de que la devuelva la función de estrategia `streamsToPublishForParticipant`.

Importante: Si `streamsToPublishForParticipant` devuelve una nueva instancia de objeto de `IVSLocalStageStream` después de una llamada a `refreshStrategy`, el estado de sonido desactivado del nuevo objeto de transmisión se aplicará al escenario. Tenga cuidado al crear nuevas instancias `IVSLocalStageStream` para asegurarse de que se mantenga el estado de sonido desactivado que se espera.

Monitoreo del estado de sonido desactivado en los contenidos multimedia del participante remoto

Cuando un participante cambia el estado de sonido desactivado de su transmisión de video o audio, se invoca la función `didChangeMutedStreams` de renderizado con una lista de las transmisiones que cambiaron. Use la propiedad `isMuted` en `IVSStageStream` para actualizar su interfaz de usuario según corresponda:

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChangeMutedStreams
 streams: [IVSStageStream]) {
    streams.forEach { stream in
        /* stream.isMuted */
    }
}
```

Creación de una configuración de escenario

Para personalizar los valores de la configuración de video de un escenario, utilice `IVSLocalStageStreamVideoConfiguration`:

```
let config = IVSLocalStageStreamVideoConfiguration()
try config.setMaxBitrate(900_000)
try config.setMinBitrate(100_000)
try config.setTargetFramerate(30)
try config.setSize(CGSize(width: 360, height: 640))
config.degradationPreference = .balanced
```

Obtención de estadísticas de WebRTC

Para obtener las estadísticas más recientes de WebRTC de una transmisión de publicación o de suscripción, utilice `requestRTCStats` en `IVSStageStream`. Cuando se complete una recopilación, recibirá estadísticas a través de `IVSStageStreamDelegate`, que se puede configurar en `IVSStageStream`. Para recopilar continuamente estadísticas de WebRTC, llame a esta función en un `Timer`.

```
func stream(_ stream: IVSStageStream, didGenerateRTCStats stats: [String : [String :
 String]]) {
    for stat in stats {
      for member in stat.value {
         print("stat \(stat.key) has member \(member.key) with value \(member.value)")
```

```
      }
   }
}
```

Obtención de los atributos de los participantes

Si especifica atributos en la solicitud del punto de conexión de `CreateParticipantToken`, puede ver los atributos en las propiedades de `IVSParticipantInfo`:

```
func stage(_ stage: IVSStage, participantDidJoin participant: IVSParticipantInfo) {
    print("ID: \(participant.participantId)")
    for attribute in participant.attributes {
        print("attribute: \(attribute.key)=\(attribute.value)")
    }
}
```

Continuación de la sesión en segundo plano

Cuando la aplicación pasa a segundo plano, puede seguir en el escenario mientras escucha el audio remoto, aunque no podrá enviar su propia imagen y audio. Es necesario que actualice la implementación `IVSStrategy` para detener la publicación y se suscriba a `.audioOnly` (o a `.none`, si corresponde):

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool {
    return false
}
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
    return .audioOnly
}
```

A continuación, haga una llamada a `stage.refreshStrategy()`.

Activación o desactivación de la codificación en capas con transmisión simultánea

Al publicar una transmisión multimedia, el SDK transmite transmisión de video de alta y baja calidad, de modo que los participantes remotos puedan suscribirse a transmisión incluso si tienen un ancho de banda de enlace descendente limitado. La codificación en capas con transmisión simultánea está activada de forma predeterminada. Puede desactivarla con `IVSSimulcastConfiguration`:

```
// Disable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = false

let cameraStream = IVSLocalStageStream(device: camera, configuration: config)

// Other Stage implementation code
```

Transmisión del escenario a un canal de IVS

Para transmitir un escenario, cree una `IVSBroadcastSession` independiente y, a continuación, siga las instrucciones habituales para la transmisión con el SDK, descritas con anterioridad. La propiedad `device` en `IVSStageStream` será un `IVSImageDevice` o `IVSAudioDevice`, tal y como se muestra en el fragmento anterior. Estos se pueden conectar a `IVSBroadcastSession.mixer` para transmitir todo el escenario en un diseño personalizable.

Si lo desea, puede componer un escenario y transmitirlo a un canal de IVS de baja latencia para llegar a un público más amplio. Consulte Habilitación de varios hosts en una transmisión de Amazon IVS en la Guía del usuario de transmisión de baja latencia.

Cómo elige iOS la resolución de la cámara y la velocidad de fotogramas

La cámara gestionada por el SDK de transmisión optimiza su resolución y velocidad de fotogramas (frames-per-secondo FPS) para minimizar la producción de calor y el consumo de energía. En esta sección se explica de qué manera se seleccionan la resolución y la velocidad de fotogramas para ayudar a las aplicaciones de host a optimizar los casos de uso.

Al crear una `IVSLocalStageStream` a una `IVSCamera`, la cámara se optimiza para una velocidad de fotogramas `IVSLocalStageStreamVideoConfiguration.targetFramerate` y una resolución de `IVSLocalStageStreamVideoConfiguration.size`. Al llamar a `IVSLocalStageStream.setConfiguration` se actualiza la cámara con valores más recientes.

Vista previa de cámara

Si crea una vista previa de una `IVSCamera` sin adjuntarla a una `IVSBroadcastSession` o `IVSStage`, la resolución predeterminada será de 1080p y la velocidad de fotogramas de 60 fps.

Transmisión de un escenario

Cuando se utiliza una `IVSBroadcastSession` para transmitir un `IVSStage`, el SDK intenta optimizar la cámara con una resolución y una velocidad de fotogramas que cumplan con los criterios de ambas sesiones.

Por ejemplo, si la configuración de transmisión está configurada para tener una velocidad de fotogramas de 15 FPS y una resolución de 1080p, mientras que el escenario tiene una velocidad de fotogramas de 30 FPS y una resolución de 720p, el SDK seleccionará una configuración de cámara con una velocidad de fotogramas de 30 FPS y una resolución de 1080p. La `IVSBroadcastSession` eliminará el resto de fotogramas de la cámara y `IVSStage` escalará la imagen de 1080p a 720p.

Si una aplicación host planea usar tanto `IVSBroadcastSession` como `IVSStage`, con una cámara, recomendamos que las propiedades de `targetFramerate` y `size` de las configuraciones respectivas coincidan. Si no coinciden, la cámara podría reconfigurase a sí misma al capturar video, lo que provocaría un breve retraso en la entrega de las muestras de video.

Si tener valores idénticos no cumple con los requisitos de uso de la aplicación host, crear primero la cámara de mayor calidad evitará que la cámara se reconfigure por sí sola cuando se agregue la sesión de menor calidad. Por ejemplo, si emite a 1080p y 30 FPS y, posteriormente, se une a un escenario configurado en 720p y 30 FPS, la cámara no se reconfigurará por sí sola y el video continuará sin interrupciones. Esto se debe a que 720p es inferior o igual a 1080p y 30 FPS es inferior o igual a 30 FPS.

Velocidades de fotogramas, resoluciones y proporciones de aspecto arbitrarias

La mayoría del hardware de las cámaras puede coincidir exactamente con los formatos habituales, como 720p a 30 FPS o 1080p a 60 FPS. Sin embargo, no se puede hacer coincidir exactamente con todos los formatos. El SDK de transmisión elige la configuración de la cámara según las siguientes reglas (en orden de prioridad):

1. El ancho y el alto de la resolución son mayores o iguales a la resolución deseada, pero dentro de esta restricción, el ancho y el alto son lo más pequeños posible.
2. La velocidad de fotogramas es mayor o igual a la velocidad de fotogramas deseada, pero dentro de esta restricción, la velocidad de fotogramas es lo más baja posible.
3. La relación de aspecto coincide con la relación de aspecto deseada.
4. Si hay varios formatos coincidentes, se utiliza el formato con el mayor campo de visión.

A continuación, se incluyen dos ejemplos:

- La aplicación host está intentando transmitir en 4K a 120 FPS. La cámara seleccionada solo admite 4K a 60 FPS o 1080p a 120 FPS. El formato seleccionado será 4K a 60 FPS, porque la regla de resolución tiene mayor prioridad que la regla de velocidad de fotogramas.
- Se solicita una resolución irregular, 1910x1070. La cámara utilizará 1920x1080. Tenga cuidado: si elige una resolución como 1921x1080, la cámara escalará hasta la siguiente resolución disponible (por ejemplo, 2592x1944), lo que supone una penalización del ancho de banda de la memoria y de la CPU.

¿Qué pasa con Android?

Android no ajusta la resolución ni la velocidad de fotogramas sobre la marcha como lo hace iOS, por lo que esto no afecta al SDK de transmisión de Android.

Problemas conocidos y soluciones alternativas

- Cambiar las rutas de audio Bluetooth puede ser impredecible. Si conecta un dispositivo nuevo a mitad de la sesión, iOS puede o no cambiar automáticamente la ruta de entrada. Además, no es posible elegir entre varios auriculares Bluetooth conectados al mismo tiempo. Esto ocurre tanto en las sesiones normales de transmisión como en las del escenario.

  Solución alternativa: si planea utilizar auriculares Bluetooth, conéctelos antes de iniciar la transmisión o el escenario y déjelos conectados durante toda la sesión.
- Los participantes que utilicen un iPhone 14, iPhone 14 Plus, iPhone 14 Pro o iPhone 14 Pro Max pueden provocar problemas de eco en otros participantes.

  Solución alternativa: los participantes que utilicen los dispositivos afectados pueden utilizar auriculares para evitar que otros participantes sufran el problema de eco.
- Cuando un participante se incorpora con un token que utiliza otro participante, la primera conexión se pierde sin que se produzca un error específico.

  Solución alternativa: ninguna.
- Hay un problema poco frecuente en el que el editor publica, pero el estado de publicación que reciben los suscriptores es `inactive`.

  Solución alternativa: pruebe a salir de la sesión y, a continuación, reincorporarse. Si el problema persiste, cree un nuevo token para el publicador.

- Cuando un participante publica o se suscribe, es posible recibir un error con el código 1400 que indica la desconexión debido a un problema de red, incluso cuando la red es estable.

  Solución alternativa: intente volver a publicar o a suscribirse.
- Durante una sesión del escenario, se puede producir un problema intermitente y poco frecuente de distorsión de audio, por lo general en llamadas de mayor duración.

  Solución alternativa: el participante con audio distorsionado puede abandonar la sesión y volver a unirse a ella o anular la publicación y volver a publicar su audio para solucionar el problema.

Control de errores

Errores fatales frente a errores no fatales

El objeto de error tiene un booleano “es grave”. Se trata de una entrada de diccionario `IVSBroadcastErrorIsFatalKey` que contiene un booleano.

En general, los errores fatales están relacionados con la conexión al servidor de etapas (o bien no se puede establecer una conexión o bien se pierde y no se puede recuperar). La aplicación debe volver a crear el escenario y volver a unirse, posiblemente con un nuevo token o cuando se recupere la conectividad del dispositivo.

Los errores no fatales suelen estar relacionados con el estado de publicación/suscripción y son gestionados por el SDK, que reintenta la operación de publicación/suscripción.

Puede comprobar esta propiedad:

```
let nsError = error as NSError
if nsError.userInfo[IVSBroadcastErrorIsFatalKey] as? Bool == true {
  // the error is fatal
}
```

Errores de unión

Token con formato incorrecto

Esto ocurre cuando el token de la etapa tiene un formato incorrecto.

El SDK lanza una excepción de Swift con el código de error = 1000 e IVS BroadcastErrorIsFatalKey = YES.

Acción: cree un token válido e intente unirse de nuevo.

Token vencido

Esto ocurre cuando el token de la etapa está caducado.

El SDK lanza una excepción de Swift con el código de error 1001 e IVS BroadcastErrorIsFatalKey = YES.

Acción: cree un token nuevo e intente unirse de nuevo.

Token no válido o revocado

Esto ocurre cuando el token de la etapa no tiene un formato incorrecto, sino que el servidor de etapas lo rechaza. Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK llama `stage(didChange connectionState, withError error)` con el código de error = 1026 e IVS BroadcastErrorIsFatalKey = YES.

Acción: cree un token válido e intente unirse de nuevo.

Errores de red durante la unión inicial

Esto ocurre cuando el SDK no puede ponerse en contacto con el servidor de etapas para establecer una conexión. Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK llama `stage(didChange connectionState, withError error)` con el código de error = 1300 e IVS BroadcastErrorIsFatalKey = YES.

Acción: espere a que se recupere la conectividad del dispositivo e intente conectarse de nuevo.

Errores de red cuando ya está conectado

Si se interrumpe la conexión de red del dispositivo, es posible que el SDK pierda la conexión con los servidores de etapas. Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK llama `stage(didChange connectionState, withError error)` con el código de error = 1300 y el BroadcastErrorIsFatalKey valor IVS = SÍ.

Acción: espere a que se recupere la conectividad del dispositivo e intente conectarse de nuevo.

Errores de publicación/suscripción

Inicial

Existen varios errores:

- MultihostSessionOfferCreationFailPublish (1020)
- MultihostSessionOfferCreationFailSubscribe (1021)
- MultihostSessionNoIceCandidates (1022)
- MultihostSessionStageAtCapacity (1024)
- SignallingSessionCannotRead (1201)
- SignallingSessionCannotSend (1202)
- SignallingSessionBadResponse (1203)

Estos se informan de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK vuelve a intentar la operación un número limitado de veces. Durante los reintentos, el estado de publicación/suscripción es `ATTEMPTING_PUBLISH`/`ATTEMPTING_SUBSCRIBE`. Si el reintento se realiza correctamente, el estado cambia a `PUBLISHED`/`SUBSCRIBED`.

El SDK llama `IVSErrorSourceDelegate:didEmitError` con el código de error correspondiente e IVS BroadcastErrorIsFatalKey = NO.

Acción: no es necesario realizar ninguna acción, ya que el SDK vuelve a intentarlo automáticamente. Si lo desea, la aplicación puede actualizar la estrategia para forzar más reintentos.

Ya está establecido, luego falla

Una publicación o una suscripción pueden fallar una vez establecidas, muy probablemente debido a un error de red. El código de error para “Se ha perdido la conexión de pares debido a un error de red” es 1400.

Esto se informa de forma asíncrona a través del renderizador de la etapa suministrado por la aplicación.

El SDK vuelve a intentar la operación de publicación/suscripción. Durante los reintentos, el estado de publicación/suscripción es `ATTEMPTING_PUBLISH`/`ATTEMPTING_SUBSCRIBE`. Si el reintento se realiza correctamente, el estado cambia a `PUBLISHED`/`SUBSCRIBED`.

El SDK llama `didEmitError` con el código de error = 1400 e IVS BroadcastErrorIsFatalKey = NO.

Acción: no es necesario realizar ninguna acción, ya que el SDK vuelve a intentarlo automáticamente. Si lo desea, la aplicación puede actualizar la estrategia para forzar más reintentos. En caso de pérdida total de conectividad, es probable que la conexión a las etapas también falle.

SDK de transmisión de IVS: orígenes de imágenes personalizados (streaming en tiempo real)

Los orígenes de entrada de imágenes personalizados permiten que una aplicación proporcione su propia entrada de imagen al SDK de transmisión, en lugar de limitarse a las cámaras predeterminadas. Un origen de imagen personalizado puede ser tan simple como una marca de agua semitransparente o una escena estática de “vuelvo enseguida”, o puede permitir que la aplicación haga un procesamiento personalizado adicional, como agregar filtros de belleza a la cámara.

Cuando utiliza una fuente de entrada de imagen personalizada para el control personalizado de la cámara (como el uso de bibliotecas de filtros de belleza que requieren acceso a la cámara), el SDK de transmisión ya no es responsable de administrar la cámara. En cambio, la aplicación es responsable de manejar correctamente el ciclo de vida de la cámara. Consulte la documentación oficial de la plataforma sobre cómo su aplicación debe administrar la cámara.

Android

Después de crear una sesión de `DeviceDiscovery`, cree un origen de entrada de imagen:

```
CustomImageSource imageSource = deviceDiscovery.createImageInputSource(new
 BroadcastConfiguration.Vec2(1280, 720));
```

Este método devuelve un `CustomImageSource`, que es una fuente de imagen respaldada por una Android Surface (Superficie) estándar. La subclase `SurfaceSource` se puede cambiar de tamaño y rotar. También puede crear un `ImagePreviewView` para mostrar una vista previa de su contenido.

Para recuperar el `Surface` subyacente :

```
Surface surface = surfaceSource.getInputSurface();
```

Este `Surface` se puede usar como búfer de salida para productores de imágenes como Camera2, OpenGL ES y otras bibliotecas. El caso de uso más simple es dibujar directamente un mapa de bits estático o un color en el lienzo de la superficie. Sin embargo, muchas bibliotecas (como las bibliotecas de filtros de belleza) proporcionan un método que permite que una aplicación especifique un `Surface` externo para la representación. Puede usar dicho método para pasar el `Surface` a la biblioteca de filtros, lo que permite que la biblioteca genere fotogramas procesados para que la sesión de transmisión los transmita.

Este `CustomImageSource` se puede encapsular en una `LocalStageStream`.
La `StageStrategy` lo puede devolver para publicarlo en un `Stage`.

iOS

Después de crear una sesión de `DeviceDiscovery`, cree un origen de entrada de imagen:

```
let customSource = broadcastSession.createImageSource(withName: "customSourceName")
```

Este método devuelve un `IVSCustomImageSource`, que es una fuente de imagen que permite que la aplicación envíe `CMSampleBuffers` manualmente. Para conocer los formatos de píxeles admitidos, consulte la referencia del SDK de transmisión de iOS; un enlace a la versión más actual se encuentra en las Notas de la versión de Amazon IVS para la última versión del SDK de transmisión.

Las muestras enviadas al origen personalizado se transmitirán en el escenario:

```
customSource.onSampleBuffer(sampleBuffer)
```

Para transmitir video, utilice este método en una devolución de llamada. Por ejemplo, si está utilizando la cámara, cada vez que se recibe un nuevo búfer de muestra de un `AVCaptureSession`, la aplicación puede reenviar el búfer de muestra a la fuente de imagen personalizada. Si lo desea, la aplicación puede aplicar más procesamiento (como un filtro de belleza) antes de enviar la muestra a la fuente de imagen personalizada.

El `IVSCustomImageSource` se puede encapsular en `IVSLocalStageStream`.
La `IVSStageStrategy` lo puede devolver para publicarlo en un `Stage`.

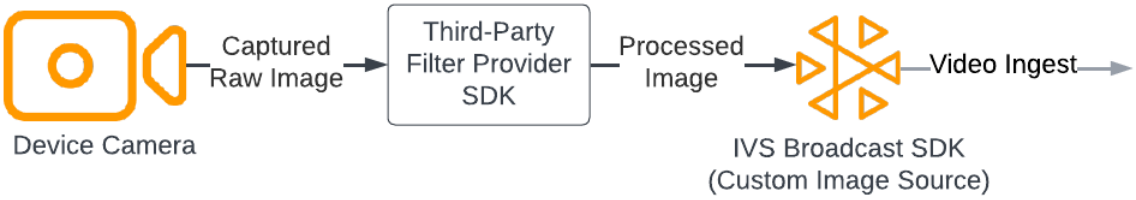
SDK de transmisión de IVS: filtros de cámara de terceros (streaming en tiempo real)

Esta guía supone que ya está familiarizado con las fuentes de imágenes personalizadas y que ha integrado el SDK de transmisión en tiempo real de IVS en su aplicación.

Los filtros de cámara permiten a los creadores de retransmisiones en directo aumentar o modificar su aspecto facial o de fondo. Esto puede aumentar la participación de los espectadores, atraerlos y mejorar la experiencia de transmisión en directo.

Integración de filtros de cámara de terceros

Puede integrar los SDK de filtros de cámara de terceros con el SDK de transmisión de IVS introduciendo la salida del SDK de filtros en una fuente de entrada de imágenes personalizada. Los orígenes de entrada de imágenes personalizados permiten que una aplicación proporcione su propia entrada de imagen al SDK de transmisión. El SDK de un proveedor de filtros externo puede gestionar el ciclo de vida de la cámara para procesar las imágenes de la cámara, aplicar un efecto de filtro y emitirlas en un formato que se pueda pasar a una fuente de imágenes personalizada.



Consulte la documentación de su proveedor de filtros externo para conocer los métodos integrados para convertir un fotograma de cámara, con el efecto de filtro aplicado a un formato que se pueda pasar a una fuente de entrada de imágenes personalizada. El proceso varía según la versión del SDK de transmisión de IVS que se utilice:

- Web: el proveedor del filtro debe poder renderizar su salida en un elemento canvas. Se puede usar el método CaptureStream para devolver un MediaStream del contenido del lienzo. El MediaStream se puede convertir en una instancia de LocalStageStream y publicarse en un escenario.
- Android: el SDK del proveedor de filtros puede renderizar un marco en un dispositivo Android `Surface` proporcionado por el SDK de transmisión de IVS o convertir el marco en un mapa de bits. Si utilizas un mapa de bits, puedes renderizarlo en la base `Surface` proporcionada por la fuente de imagen personalizada desbloqueándolo y escribiéndolo en un lienzo.
- iOS: el SDK de un proveedor de filtros externo debe proporcionar un marco de cámara con un efecto de filtro aplicado como `CMSampleBuffer`. Consulte la documentación del SDK de su

proveedor de filtros externo para obtener información sobre cómo obtener un `CMSampleBuffer` como resultado final después de procesar una imagen de cámara.

BytePlus

Android

Instale y configure el SDK de efectos de BytePlus

Consulte la Guía de acceso de BytePlus para Android para obtener detalles sobre cómo instalar, inicializar y configurar el SDK de BytePlus Effects.

Configure la Fuente de imagen personalizada

Tras inicializar el SDK, alimente los fotogramas de cámara procesados con un efecto de filtro aplicado a una fuente de entrada de imágenes personalizadas. Para ello, cree una instancia de un objeto `DeviceDiscovery` y cree una fuente de imagen personalizada. Observe que cuando utiliza una fuente de entrada de imagen personalizada para el control personalizado de la cámara, el SDK de transmisión ya no es responsable de administrar la cámara. En cambio, la aplicación es responsable de manejar correctamente el ciclo de vida de la cámara.

Java

```
var deviceDiscovery = DeviceDiscovery(applicationContext)
var customSource = deviceDiscovery.createImageInputSource( BroadcastConfiguration.Vec2(
720F, 1280F
))
var surface: Surface = customSource.inputSurface
var filterStream = ImageLocalStageStream(customSource)
```

Convierta la salida en un mapa de bits y la alimente en una fuente de entrada de imagen personalizada

Para permitir que los fotogramas de cámara con un efecto de filtro aplicado desde el SDK de efectos de BytePlus se reenvíen directamente al SDK de transmisión de IVS, convierta la salida de una textura del SDK de BytePlus Effects en un mapa de bits. Cuando se procesa una imagen, el `onDrawFrame()` SDK invoca el método. El `onDrawFrame()` método es un método público de la interfaz GLSurfaceView.Renderer de Android. En la aplicación de ejemplo para Android proporcionada por BytePlus, este método es llamado en todos los fotogramas de la cámara

y genera una textura. Al mismo tiempo, puedes complementar el `onDrawFrame()` método con la lógica para convertir esta textura en un mapa de bits y enviarla a una fuente de entrada de imágenes personalizada. Como se muestra en el siguiente ejemplo de código, utilice el `transferTextureToBitmap` método proporcionado por el SDK de BytePlus para realizar esta conversión. Este método lo proporciona la biblioteca com.bytedance.labcv.core.util.ImageUtil del SDK de BytePlus Effects, como se muestra en el siguiente ejemplo de código. Luego, puede renderizar en el Android subyacente `Surface` de `CustomImageSource` escribiendo el mapa de bits resultante en el lienzo de Surface. Muchas invocaciones sucesivas de `onDrawFrame()` dan como resultado una secuencia de mapas de bits y, cuando se combinan, crean una secuencia de vídeo.

Java

```
import com.bytedance.labcv.core.util.ImageUtil;
...
protected ImageUtil imageUtility;
...


@Override
public void onDrawFrame(GL10 gl10) {
  ...
  // Convert BytePlus output to a Bitmap
  Bitmap outputBt = imageUtility.transferTextureToBitmap(output.getTexture(),ByteEffect

  Constants.TextureFormat.Texture2D,output.getWidth(), output.getHeight());

  canvas = surface.lockCanvas(null);
  canvas.drawBitmap(outputBt, 0f, 0f, null);
  surface.unlockCanvasAndPost(canvas);
```

DeepAR

Android

Consulta la Guía de integración de Android de DeepAR para obtener detalles sobre cómo integrar el SDK de DeepAR con el SDK de retransmisión de Android IVS.

iOS

Consulte la Guía de integración de iOS de DeepAR para obtener más información sobre cómo integrar el SDK de DeepAR con el SDK de transmisión IVS para iOS.

Snap

Web

En esta sección se presupone que ya está familiarizado con la publicación de vídeos y la suscripción a ellos mediante el SDK de transmisión web.

Para integrar el SDK del kit de cámara de Snap con el SDK de streaming Web en tiempo real de IVS, debes:

1. Instala el SDK y el Webpack del kit de cámara. (Nuestro ejemplo usa Webpack como paquete, pero puedes usar cualquier paquete de tu elección).
2. Create `index.html`.
3. Agrega elementos de configuración.
4. Muestra y configura los participantes.
5. Muestra las cámaras y los micrófonos conectados.
6. Crea una sesión de Camera Kit.
7. Busca y aplica una lente.
8. Renderice el resultado de una sesión de Camera Kit en un lienzo.
9. Proporcione a Camera Kit una fuente multimedia para renderizar y publicar un`LocalStageStream`.
10. Cree un archivo de configuración de Webpack.

A continuación, se describe cada uno de estos pasos.

Instale el kit de cámara, el SDK y el paquete web

```
npm i @snap/camera-kit webpack webpack-cli
```

Cree el archivo index.html

A continuación, cree la plantilla HTML reutilizable e importe el SDK de transmisión para web como una etiqueta script. En el código siguiente, asegúrese de reemplazarlo `<SDK version>` por la versión del SDK de difusión que esté utilizando.

JavaScript

```
<!--
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */
-->
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <title>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</title>

  <!-- Fonts and Styling -->
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Roboto:300,300italic,700,700italic" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/
normalize.css" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/milligram/1.4.1/
milligram.css" />
  <link rel="stylesheet" href="./index.css" />

  <!-- Stages in Broadcast SDK -->
  <script src="https://web-broadcast.live-video.net/<SDK version>/amazon-ivs-web-
broadcast.js"></script>
</head>

<body>
  <!-- Introduction -->
  <header>
    <h1>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</h1>

    <p>This sample is used to demonstrate basic HTML / JS usage. <b><a href="https://
docs.aws.amazon.com/ivs/latest/userguide/multiple-hosts.html">Use the AWS CLI</
a></b> to create a <b>Stage</b> and a corresponding <b>ParticipantToken</b>.
 Multiple participants can load this page and put in their own tokens. You can <b><a
 href="https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#glossary"
 target="_blank">read more about stages in our public docs.</a></b></p>
  </header>
  <hr />
```

```
  <!-- Setup Controls -->

  <!-- Local Participant -->

  <hr style="margin-top: 5rem"/>

  <!-- Remote Participants -->

  <!-- Load all Desired Scripts -->

</body>

</html>
```

Agregue elementos de configuración

Cree el código HTML para seleccionar una cámara y un micrófono y especificar un token de participante:

JavaScript

```
<!-- Setup Controls -->
  <div class="row">
    <div class="column">
      <label for="video-devices">Select Camera</label>
      <select disabled id="video-devices">
        <option selected disabled>Choose Option</option>
      </select>
    </div>
    <div class="column">
      <label for="audio-devices">Select Microphone</label>
      <select disabled id="audio-devices">
        <option selected disabled>Choose Option</option>
      </select>
    </div>
    <div class="column">
      <label for="token">Participant Token</label>
      <input type="text" id="token" name="token" />
    </div>
    <div class="column" style="display: flex; margin-top: 1.5rem">
      <button class="button" style="margin: auto; width: 100%" id="join-button">Join
 Stage</button>
    </div>
    <div class="column" style="display: flex; margin-top: 1.5rem">
```

```
      <button class="button" style="margin: auto; width: 100%" id="leave-button">Leave
 Stage</button>
    </div>
  </div>
```

Añada un código HTML adicional debajo para mostrar las imágenes de las cámaras de los participantes locales y remotos:

JavaScript

```
 <!-- Local Participant -->
<div class="row local-container">
    <canvas id="canvas"></canvas>

    <div class="column" id="local-media"></div>
    <div class="static-controls hidden" id="local-controls">
      <button class="button" id="mic-control">Mute Mic</button>
      <button class="button" id="camera-control">Mute Camera</button>
    </div>
  </div>


  <hr style="margin-top: 5rem"/>

  <!-- Remote Participants -->
  <div class="row">
    <div id="remote-media"></div>
  </div>
```

Carga lógica adicional, incluidos los métodos de ayuda para configurar la cámara y el archivo JavaScript incluido. (Más adelante en esta sección, creará estos archivos JavaScript y los agrupará en un solo archivo para poder importar Camera Kit como un módulo. El archivo JavaScript incluido contendrá la lógica para configurar el kit de cámara, aplicar un objetivo y publicar la imagen de la cámara con un objetivo aplicado a un escenario).

JavaScript

```
<!-- Load all Desired Scripts -->
  <script src="./helpers.js"></script>
  <script src="./media-devices.js"></script>
  <!-- <script type="module" src="./stages-simple.js"></script> -->
  <script src="./dist/bundle.js"></script>
```

Mostrar y configurar los participantes

A continuación, cree`helpers.js`, que contiene métodos auxiliares que utilizará para mostrar y configurar los participantes:

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

function setupParticipant({ isLocal, id }) {
  const groupId = isLocal ? 'local-media' : 'remote-media';
  const groupContainer = document.getElementById(groupId);

  const participantContainerId = isLocal ? 'local' : id;
  const participantContainer = createContainer(participantContainerId);
  const videoEl = createVideoEl(participantContainerId);

  participantContainer.appendChild(videoEl);
  groupContainer.appendChild(participantContainer);

  return videoEl;
}

function teardownParticipant({ isLocal, id }) {
  const groupId = isLocal ? 'local-media' : 'remote-media';
  const groupContainer = document.getElementById(groupId);
  const participantContainerId = isLocal ? 'local' : id;

  const participantDiv = document.getElementById(
    participantContainerId + '-container'
  );
  if (!participantDiv) {
    return;
  }
  groupContainer.removeChild(participantDiv);
}

function createVideoEl(id) {
  const videoEl = document.createElement('video');
  videoEl.id = id;
  videoEl.autoplay = true;
  videoEl.playsInline = true;
  videoEl.srcObject = new MediaStream();
```

```
  return videoEl;
}

function createContainer(id) {
  const participantContainer = document.createElement('div');
  participantContainer.classList = 'participant-container';
  participantContainer.id = id + '-container';

  return participantContainer;
}
```

Muestra las cámaras y los micrófonos conectados

A continuación, creemedia-devices.js, que contiene métodos auxiliares para mostrar las cámaras y los micrófonos conectados a su dispositivo:

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

/**
 * Returns an initial list of devices populated on the page selects
 */
async function initializeDeviceSelect() {
  const videoSelectEl = document.getElementById('video-devices');
  videoSelectEl.disabled = false;

  const { videoDevices, audioDevices } = await getDevices();
  videoDevices.forEach((device, index) => {
    videoSelectEl.options[index] = new Option(device.label, device.deviceId);
  });

  const audioSelectEl = document.getElementById('audio-devices');

  audioSelectEl.disabled = false;
  audioDevices.forEach((device, index) => {
    audioSelectEl.options[index] = new Option(device.label, device.deviceId);
  });
}

/**
 * Returns all devices available on the current device
```

```
 */
async function getDevices() {
  // Prevents issues on Safari/FF so devices are not blank
  await navigator.mediaDevices.getUserMedia({ video: true, audio: true });

  const devices = await navigator.mediaDevices.enumerateDevices();
  // Get all video devices
  const videoDevices = devices.filter((d) => d.kind === 'videoinput');
  if (!videoDevices.length) {
    console.error('No video devices found.');
  }

  // Get all audio devices
  const audioDevices = devices.filter((d) => d.kind === 'audioinput');
  if (!audioDevices.length) {
    console.error('No audio devices found.');
  }

  return { videoDevices, audioDevices };
}

async function getCamera(deviceId) {
  // Use Max Width and Height
  return navigator.mediaDevices.getUserMedia({
    video: {
      deviceId: deviceId ? { exact: deviceId } : null,
    },
    audio: false,
  });
}

async function getMic(deviceId) {
  return navigator.mediaDevices.getUserMedia({
    video: false,
    audio: {
      deviceId: deviceId ? { exact: deviceId } : null,
    },
  });
}
```

Cree una sesión de kit de cámara

Create`stages.js`, que contiene la lógica para aplicar una lente a la transmisión de la cámara y publicar la transmisión en un escenario. En la primera parte de este archivo, importamos el SDK de transmisión y el SDK web de Camera Kit e inicializamos las variables que usaremos con cada SDK. Para crear una sesión de Camera Kit, llamamos `createSession` después de iniciar el Camera Kit Web SDK. Tenga en cuenta que un objeto de elemento canvas se pasa a una sesión; esto le indica a Camera Kit que lo renderice en ese lienzo.

Java

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

// All helpers are expose on 'media-devices.js' and 'dom.js'
// const { setupParticipant } = window;
// const { initializeDeviceSelect, getCamera, getMic } = window;
// require('./helpers.js');
// require('./media-devices.js');

const {
  Stage,
  LocalStageStream,
  SubscribeType,
  StageEvents,
  ConnectionState,
  StreamType,
} = IVSBroadcastClient;

import {
  bootstrapCameraKit,
  createMediaStreamSource,
  Transform2D,
} from '@snap/camera-kit';

let cameraButton = document.getElementById('camera-control');
let micButton = document.getElementById('mic-control');
let joinButton = document.getElementById('join-button');
let leaveButton = document.getElementById('leave-button');

let controls = document.getElementById('local-controls');
let videoDevicesList = document.getElementById('video-devices');
let audioDevicesList = document.getElementById('audio-devices');
```

```
// Stage management
let stage;
let joining = false;
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
let micStageStream;

const liveRenderTarget = document.getElementById('canvas');

const init = async () => {
  await initializeDeviceSelect();

  const cameraKit = await bootstrapCameraKit({
    apiToken: INSERT_API_TOKEN_HERE,
  });

  const session = await cameraKit.createSession({ liveRenderTarget });
```

Busca y aplica una lente

Para buscar sus lentes, introduzca su ID de grupo de lentes, que puede encontrar en el portal para desarrolladores de kits de cámara. En este ejemplo, lo simplificamos aplicando la primera lente de la matriz de lentes que se devuelve.

JavaScript

```
const { lenses } = await cameraKit.lensRepository.loadLensGroups([
    INSERT_LENS_GROUP_ID_HERE,
  ]);

  session.applyLens(lenses[0]);
```

Renderice el resultado de una sesión del kit de cámara en un lienzo

Utilice el método CaptureStream para devolver parte del contenido `MediaStream` del lienzo. El lienzo contendrá una secuencia de vídeo de la imagen de la cámara con una lente aplicada. Además, añada detectores de eventos como botones para silenciar la cámara y el micrófono, así como detectores de eventos para entrar y salir del escenario. En el oyente de eventos para unirse

a un escenario, pasamos una sesión del kit de cámara y la `MediaStream` del lienzo para poder publicarla en un escenario.

JavaScript

```
const snapStream = liveRenderTarget.captureStream();

  cameraButton.addEventListener('click', () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? 'Show Camera' : 'Hide Camera';
  });

  micButton.addEventListener('click', () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? 'Unmute Mic' : 'Mute Mic';
  });

  joinButton.addEventListener('click', () => {
    joinStage(session, snapStream);
  });

  leaveButton.addEventListener('click', () => {
    leaveStage();
  });
};
```

Proporcione a Camera Kit una fuente multimedia para renderizar y publique un LocalStageStream

Para publicar una transmisión de vídeo con una lente aplicada, cree una función llamada `setCameraKitSource` a transferir lo `MediaStream` capturado anteriormente desde el lienzo. La `MediaStream` desde el lienzo no sirve de nada por el momento porque aún no hemos incorporado nuestra cámara local. Podemos incorporar la señal de nuestra cámara local llamando `getCamera` al método auxiliar y asignándolo a `localCamera`. Luego podemos pasar la señal de nuestra cámara local (via`localCamera`) y el objeto de sesión a `setCameraKitSource`. La `setCameraKitSource` función convierte la señal de nuestra cámara local en una fuente de contenido multimedia para CameraKit con solo una llamada `createMediaStreamSource`. La fuente multimedia `CameraKit` se transforma para reflejar la cámara frontal. El efecto Lente se aplica a la fuente multimedia y se renderiza en el lienzo de salida mediante una llamada`session.play()`.

Ahora que la lente está aplicada a lo `MediaStream` capturado desde el lienzo, podemos proceder a publicarlo en un escenario. Para ello, creamos un `LocalStageStream` con las pistas de vídeo del`MediaStream`. Luego, se `LocalStageStream` puede pasar una instancia de a `StageStrategy` para publicarla.

JavaScript

```
async function setCameraKitSource(session, mediaStream) {
  const source = createMediaStreamSource(mediaStream);
  await session.setSource(source);
  source.setTransform(Transform2D.MirrorX);
  session.play();
}

const joinStage = async (session, snapStream) => {
  if (connected || joining) {
    return;
  }
  joining = true;

  const token = document.getElementById('token').value;

  if (!token) {
    window.alert('Please enter a participant token');
    joining = false;
    return;
  }

  // Retrieve the User Media currently set on the page
  localCamera = await getCamera(videoDevicesList.value);
  localMic = await getMic(audioDevicesList.value);
  await setCameraKitSource(session, localCamera);
  // Create StageStreams for Audio and Video
  // cameraStageStream = new LocalStageStream(localCamera.getVideoTracks()[0]);
  cameraStageStream = new LocalStageStream(snapStream.getVideoTracks()[0]);
  micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);

  const strategy = {
    stageStreamsToPublish() {
      return [cameraStageStream, micStageStream];
    },
    shouldPublishParticipant() {
      return true;
```

```
    },
    shouldSubscribeToParticipant() {
      return SubscribeType.AUDIO_VIDEO;
    },
  };
```

El código restante que aparece a continuación sirve para crear y gestionar nuestro escenario:

JavaScript

```
stage = new Stage(token, strategy);

  // Other available events:
  // https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#events

  stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
    connected = state === ConnectionState.CONNECTED;

    if (connected) {
      joining = false;
      controls.classList.remove('hidden');
    } else {
      controls.classList.add('hidden');
    }
  });

  stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
    console.log('Participant Joined:', participant);
  });

  stage.on(
    StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED,
    (participant, streams) => {
      console.log('Participant Media Added: ', participant, streams);

      let streamsToDisplay = streams;

      if (participant.isLocal) {
        // Ensure to exclude local audio streams, otherwise echo will occur
        streamsToDisplay = streams.filter(
          (stream) => stream.streamType === StreamType.VIDEO
        );
      }
```

```
      const videoEl = setupParticipant(participant);
      streamsToDisplay.forEach((stream) =>
        videoEl.srcObject.addTrack(stream.mediaStreamTrack)
      );
    }
  );

  stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {
    console.log('Participant Left: ', participant);
    teardownParticipant(participant);
  });

  try {
    await stage.join();
  } catch (err) {
    joining = false;
    connected = false;
    console.error(err.message);
  }
};

const leaveStage = async () => {
  stage.leave();

  joining = false;
  connected = false;

  cameraButton.innerText = 'Hide Camera';
  micButton.innerText = 'Mute Mic';
  controls.classList.add('hidden');
};

init();
```

Crear un archivo de configuración de Webpack

Abra `webpack.config.js` y agregue el siguiente código. Esto agrupa la lógica anterior para que pueda usar la declaración de importación para usar Camera Kit.

JavaScript

```
const path = require('path');
module.exports = {
```

```
  entry: ['./stage.js'],
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

Por último, ejecute `npm run build` para empaquetar su JavaScript como se define en el archivo de configuración de Webpack. A continuación, podrá servir HTML y JavaScript desde un servidor web. Por ejemplo, puedes usar el servidor HTTP de Python y abrir localhost:8000 para ver el resultado:

```
# Run this from the command line and the directory containing index.html
python3 -m http.server -d ./
```

Android

Para integrar el SDK del Camera Kit de Snap con el SDK de retransmisión para Android de IVS, debes instalar el SDK del Camera Kit, inicializar una sesión del Camera Kit, aplicar un objetivo y enviar el resultado de la sesión del Camera Kit a la fuente de entrada de la imagen personalizada.

Para instalar el SDK del kit de cámara, añade lo siguiente al archivo de tu módulo`build.gradle`. `$cameraKitVersion`Sustitúyalo por la última versión del SDK del Camera Kit.

Java

```
implementation "com.snap.camerakit:camerakit:$cameraKitVersion"
```

Inicialice y obtenga un `cameraKitSession`. Camera Kit también proporciona un práctico contenedor para las API CameraX de Android, por lo que no tienes que escribir una lógica complicada para usar CameraX con Camera Kit. Puedes usar el `CameraXImageProcessorSource` objeto como fuente para ImageProcessor, que te permite iniciar fotogramas de streaming con vistas previas de la cámara.

Java

```
protected void onCreate(@Nullable Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       setContentView(R.layout.activity_main);
```

```
        // Camera Kit support implementation of ImageProcessor that is backed by
 CameraX library:
        // https://developer.android.com/training/camerax
        CameraXImageProcessorSource imageProcessorSource = new
 CameraXImageProcessorSource(
            this /*context*/, this /*lifecycleOwner*/
        );
        imageProcessorSource.startPreview(true /*cameraFacingFront*/);

        cameraKitSession = Sessions.newBuilder(this)
                .imageProcessorSource(imageProcessorSource)
                .attachTo(findViewById(R.id.camerakit_stub))
                .build();
    }
```

Busca y aplica lentes

Puede configurar los lentes y su orden en el carrusel del portal para desarrolladores de kits de cámara:

Java

```
// Fetch lenses from repository and apply them
 // Replace LENS_GROUP_ID with Lens Group ID from https://camera-kit.snapchat.com
cameraKitSession.getLenses().getRepository().get(new Available(LENS_GROUP_ID),
 available -> {
     Log.d(TAG, "Available lenses: " + available);
     Lenses.whenHasFirst(available, lens ->
 cameraKitSession.getLenses().getProcessor().apply(lens, result -> {
          Log.d(TAG,  "Apply lens [" + lens + "] success: " + result);
      }));
});
```

Para transmitir, envíe los fotogramas procesados a la base de una `Surface` fuente de imagen personalizada. Usa un `DeviceDiscovery` objeto y crea un `CustomImageSource` para devolver un`SurfaceSource`. A continuación, puede renderizar el resultado de una `CameraKit` sesión en el subyacente `Surface` proporcionado por el`SurfaceSource`.

Java

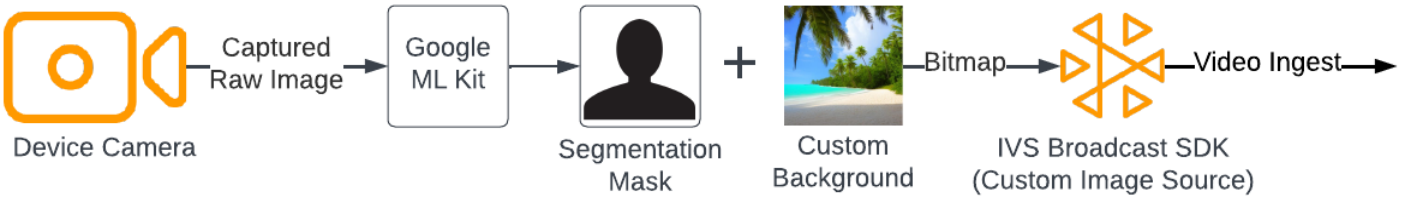
```
val publishStreams = ArrayList<LocalStageStream>()

val deviceDiscovery = DeviceDiscovery(applicationContext)
```

```
val customSource =
 deviceDiscovery.createImageInputSource(BroadcastConfiguration.Vec2(720f, 1280f))

cameraKitSession.processor.connectOutput(outputFrom(customSource.inputSurface))
val customStream = ImageLocalStageStream(customSource)

// After rendering the output from a Camera Kit session to the Surface, you can
// then return it as a LocalStageStream to be published by the Broadcast SDK
val customStream: ImageLocalStageStream = ImageLocalStageStream(surfaceSource)
publishStreams.add(customStream)

@Override
fun stageStreamsToPublishForParticipant(stage: Stage, participantInfo:
 ParticipantInfo): List<LocalStageStream> = publishStreams
```

Reemplazo de fondo

El reemplazo del fondo es un tipo de filtro de cámara que permite a los creadores de transmisiones en directo cambiar sus fondos. Como se muestra en el diagrama siguiente, reemplazar el fondo implica:

1. Obtener una imagen de cámara a partir de la transmisión de la cámara en directo.
2. Segmentarla en componentes de primer plano y segundo plano con el kit ML de Google.
3. Combinar la máscara de segmentación resultante con una imagen de fondo personalizada.
4. Pasarla a una fuente de imagen personalizada para su transmisión.



Web

En esta sección se presupone que ya está familiarizado con la publicación de vídeos y la suscripción a ellos mediante el SDK de transmisión web.

Para sustituir el fondo de una transmisión en directo por una imagen personalizada, utilice el modelo de segmentación de selfies con MediaPipe Image Segmenter. Se trata de un modelo de aprendizaje automático que identifica qué píxeles del fotograma de vídeo están en primer plano o en segundo

plano. Puede utilizar los resultados del modelo para sustituir el fondo de una transmisión en directo copiando los píxeles de primer plano de la transmisión de vídeo a una imagen personalizada que represente el nuevo fondo.

Para integrar el reemplazo del fondo de pantalla con el SDK de transmisión web en tiempo real de IVS, debe:

1. Instale MediaPipe y Webpack. (Nuestro ejemplo usa Webpack como paquete, pero puede usar cualquier paquete de su elección).
2. Create `index.html`.
3. Agregue elementos multimedia.
4. Añada una etiqueta de script.
5. Create `app.js`.
6. Carga una imagen de fondo personalizada.
7. Cree una instancia de `ImageSegmenter`.
8. Renderiza la transmisión de vídeo en un lienzo.
9. Cree una lógica de reemplazo de fondo.
10. Cree el archivo de configuración de Webpack.
11. Agrupe su archivo JavaScript.

Instale MediaPipe y Webpack

Para empezar, instale `@mediapipe/tasks-vision` y los `webpack` paquetes npm. El siguiente ejemplo usa Webpack como un empaquetador de JavaScript; puedes usar un empaquetador diferente si lo prefieres.

JavaScript

```
npm i @mediapipe/tasks-vision webpack webpack-cli
```

Asegúrate de actualizar también tu script `package.json` de compilación para `webpack` especificarlo:

JavaScript

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
```

```
    "build": "webpack"
  },
```

Cree el archivo index.html

A continuación, cree la plantilla HTML reutilizable e importe el SDK de transmisión para web como una etiqueta script. En el código siguiente, asegúrese de <SDK version> reemplazarlo por la versión del SDK de difusión que esté utilizando.

JavaScript

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Import the SDK -->
  <script src="https://web-broadcast.live-video.net/<SDK version>/amazon-ivs-web-
broadcast.js"></script>
</head>

<body>

</body>
</html>
```

Agregue elementos multimedia

A continuación, añade un elemento de vídeo y dos elementos de lienzo dentro de la etiqueta corporal. El elemento de vídeo contendrá las imágenes de la cámara en directo y se utilizará como entrada para el segmentador de imágenes de MediaPipe. El primer elemento canvas se utilizará para obtener una vista previa de la transmisión que se emitirá. El segundo elemento de lienzo se utilizará para renderizar la imagen personalizada que se utilizará como fondo. Como el segundo lienzo con la imagen personalizada solo se usa como fuente para copiar píxeles mediante programación al lienzo final, está oculto a la vista.

JavaScript

```
<div class="row local-container">
```

```
      <video id="webcam" autoplay style="display: none"></video>
    </div>
    <div class="row local-container">
      <canvas id="canvas" width="640px" height="480px"></canvas>

      <div class="column" id="local-media"></div>
      <div class="static-controls hidden" id="local-controls">
        <button class="button" id="mic-control">Mute Mic</button>
        <button class="button" id="camera-control">Mute Camera</button>
      </div>
    </div>
    <div class="row local-container">
      <canvas id="background" width="640px" height="480px" style="display: none"></
canvas>
    </div>
```

Agregue una etiqueta de script

Agregue una etiqueta de script para cargar un archivo JavaScript incluido que contendrá el código para reemplazar el fondo y publicarlo en un escenario:

```
<script src="./dist/bundle.js"></script>
```

Creación de app.js

A continuación, cree un archivo JavaScript para obtener los objetos de elemento para los elementos de lienzo y vídeo que se crearon en la página HTML. Importa los módulos `ImageSegmenter` y `FilesetResolver`. El `ImageSegmenter` módulo se utilizará para realizar la tarea de segmentación.

JavaScript

```
const canvasElement = document.getElementById("canvas");
const background = document.getElementById("background");
const canvasCtx = canvasElement.getContext("2d");
const backgroundCtx = background.getContext("2d");
const video = document.getElementById("webcam");

import { ImageSegmenter, FilesetResolver } from "@mediapipe/tasks-vision";
```

A continuación, cree una función llamada `init()` para recuperar el MediaStream de la cámara del usuario e invoque una función de devolución de llamada cada vez que termine de cargarse el

fotograma de la cámara. Añada detectores de eventos para que los botones se unan y salgan de un escenario.

Tenga en cuenta que cuando nos unimos a un escenario, pasamos una variable llamada`segmentationStream`. Se trata de una secuencia de vídeo capturada desde un elemento de lienzo, que contiene una imagen de primer plano superpuesta a la imagen personalizada que representa el fondo. Más adelante, esta transmisión personalizada se utilizará para crear una instancia de a`LocalStageStream`, que se podrá publicar en un escenario.

JavaScript

```
const init = async () => {
  await initializeDeviceSelect();

  cameraButton.addEventListener("click", () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? "Show Camera" : "Hide Camera";
  });

  micButton.addEventListener("click", () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? "Unmute Mic" : "Mute Mic";
  });

  localCamera = await getCamera(videoDevicesList.value);
  const segmentationStream = canvasElement.captureStream();

  joinButton.addEventListener("click", () => {
    joinStage(segmentationStream);
  });

  leaveButton.addEventListener("click", () => {
    leaveStage();
  });
};
```

Cargue una imagen de fondo personalizada

En la parte inferior de la `init` función, añade código para llamar a una función llamada`initBackgroundCanvas`, que carga una imagen personalizada de un archivo local y la

renderiza en un lienzo. Definiremos esta función en el siguiente paso. Asigne lo `MediaStream` recuperado de la cámara del usuario al objeto de vídeo. Más tarde, este objeto de vídeo se pasará al segmentador de imágenes. Además, configure una función `renderVideoToCanvas` denominada función de devolución de llamada para que se invoque cada vez que un fotograma de vídeo termine de cargarse. Definiremos esta función en un paso posterior.

JavaScript

```
initBackgroundCanvas();

  video.srcObject = localCamera;
  video.addEventListener("loadeddata", renderVideoToCanvas);
```

Vamos a implementar la `initBackgroundCanvas` función, que carga una imagen desde un archivo local. En este ejemplo, utilizaremos una imagen de una playa como fondo personalizado. El lienzo que contiene la imagen personalizada se ocultará de la pantalla, ya que lo combinará con los píxeles del primer plano del elemento del lienzo que contiene la imagen de la cámara.

JavaScript

```
const initBackgroundCanvas = () => {
  let img = new Image();
  img.src = "beach.jpg";

  img.onload = () => {
    backgroundCtx.clearRect(0, 0, canvas.width, canvas.height);
    backgroundCtx.drawImage(img, 0, 0);
  };
};
```

Creación de una instancia de ImageSegmenter

A continuación, cree una instancia de`ImageSegmenter`, que segmentará la imagen y devolverá el resultado en forma de máscara. Al crear una instancia de una`ImageSegmenter`, utilizarás el modelo de segmentación de selfies.

JavaScript

```
const createImageSegmenter = async () => {
  const audio = await FilesetResolver.forVisionTasks("https://cdn.jsdelivr.net/npm/
@mediapipe/tasks-vision@0.10.2/wasm");
```

```
  imageSegmenter = await ImageSegmenter.createFromOptions(audio, {
    baseOptions: {
      modelAssetPath: "https://storage.googleapis.com/mediapipe-models/image_segmenter/
selfie_segmenter/float16/latest/selfie_segmenter.tflite",
      delegate: "GPU",
    },
    runningMode: "VIDEO",
    outputCategoryMask: true,
  });
};
```

Renderice la transmisión de vídeo en un lienzo

A continuación, cree la función que renderice la transmisión de vídeo al otro elemento del lienzo. Necesitamos renderizar la transmisión de vídeo en un lienzo para poder extraer los píxeles del primer plano utilizando la API Canvas 2D. Mientras lo hacemos, también pasaremos un fotograma de vídeo a nuestra instancia`ImageSegmenter`, utilizando el método SegmentForVideo para segmentar el primer plano del fondo del fotograma de vídeo. Cuando el método SegmentForVideo regresa, invoca nuestra función de devolución de llamada personalizada, `replaceBackground` , para reemplazar el fondo.

JavaScript

```
const renderVideoToCanvas = async () => {
  if (video.currentTime === lastWebcamTime) {
    window.requestAnimationFrame(renderVideoToCanvas);
    return;
  }
  lastWebcamTime = video.currentTime;
  canvasCtx.drawImage(video, 0, 0, video.videoWidth, video.videoHeight);

  if (imageSegmenter === undefined) {
    return;
  }

  let startTimeMs = performance.now();

  imageSegmenter.segmentForVideo(video, startTimeMs, replaceBackground);
};
```

Cree una lógica de reemplazo en segundo plano

Cree la `replaceBackground` función, que fusiona la imagen de fondo personalizada con el primer plano de la transmisión de la cámara para reemplazar el fondo. La función recupera primero los datos de píxeles subyacentes de la imagen de fondo personalizada y la transmisión de vídeo de los dos elementos del lienzo creados anteriormente. A continuación, recorre en iteración la máscara proporcionada por`ImageSegmenter`, que indica qué píxeles están en primer plano. A medida que recorre la máscara, copia de forma selectiva los píxeles que contienen la imagen de la cámara del usuario en los datos de píxeles de fondo correspondientes. Una vez hecho esto, convierte los datos de píxeles finales con el primer plano copiado en el fondo y los dibuja en un lienzo.

JavaScript

```
function replaceBackground(result) {
  let imageData = canvasCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  let backgroundData = backgroundCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  const mask = result.categoryMask.getAsFloat32Array();
  let j = 0;

  for (let i = 0; i < mask.length; ++i) {
    const maskVal = Math.round(mask[i] * 255.0);

    j += 4;
  // Only copy pixels on to the background image if the mask indicates they are in the
 foreground
    if (maskVal < 255) {
      backgroundData[j] = imageData[j];
      backgroundData[j + 1] = imageData[j + 1];
      backgroundData[j + 2] = imageData[j + 2];
      backgroundData[j + 3] = imageData[j + 3];
    }
  }

 // Convert the pixel data to a format suitable to be drawn to a canvas
  const uint8Array = new Uint8ClampedArray(backgroundData.buffer);
  const dataNew = new ImageData(uint8Array, video.videoWidth, video.videoHeight);
  canvasCtx.putImageData(dataNew, 0, 0);
  window.requestAnimationFrame(renderVideoToCanvas);
}
```

Como referencia, aquí está el `app.js` archivo completo que contiene toda la lógica anterior:

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

// All helpers are expose on 'media-devices.js' and 'dom.js'
const { setupParticipant } = window;

const { Stage, LocalStageStream, SubscribeType, StageEvents, ConnectionState,
 StreamType } = IVSBroadcastClient;
const canvasElement = document.getElementById("canvas");
const background = document.getElementById("background");
const canvasCtx = canvasElement.getContext("2d");
const backgroundCtx = background.getContext("2d");
const video = document.getElementById("webcam");

import { ImageSegmenter, FilesetResolver } from "@mediapipe/tasks-vision";

let cameraButton = document.getElementById("camera-control");
let micButton = document.getElementById("mic-control");
let joinButton = document.getElementById("join-button");
let leaveButton = document.getElementById("leave-button");

let controls = document.getElementById("local-controls");
let audioDevicesList = document.getElementById("audio-devices");
let videoDevicesList = document.getElementById("video-devices");

// Stage management
let stage;
let joining = false;
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
let micStageStream;
let imageSegmenter;
let lastWebcamTime = -1;

const init = async () => {
  await initializeDeviceSelect();

  cameraButton.addEventListener("click", () => {
```

```
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? "Show Camera" : "Hide Camera";
  });

  micButton.addEventListener("click", () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? "Unmute Mic" : "Mute Mic";
  });

  localCamera = await getCamera(videoDevicesList.value);
  const segmentationStream = canvasElement.captureStream();

  joinButton.addEventListener("click", () => {
    joinStage(segmentationStream);
  });

  leaveButton.addEventListener("click", () => {
    leaveStage();
  });

  initBackgroundCanvas();

  video.srcObject = localCamera;
  video.addEventListener("loadeddata", renderVideoToCanvas);
};

const joinStage = async (segmentationStream) => {
  if (connected || joining) {
    return;
  }
  joining = true;

  const token = document.getElementById("token").value;

  if (!token) {
    window.alert("Please enter a participant token");
    joining = false;
    return;
  }

  // Retrieve the User Media currently set on the page
  localMic = await getMic(audioDevicesList.value);
```

```
  cameraStageStream = new LocalStageStream(segmentationStream.getVideoTracks()[0]);
  micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);

  const strategy = {
    stageStreamsToPublish() {
      return [cameraStageStream, micStageStream];
    },
    shouldPublishParticipant() {
      return true;
    },
    shouldSubscribeToParticipant() {
      return SubscribeType.AUDIO_VIDEO;
    },
  };

  stage = new Stage(token, strategy);

  // Other available events:
  // https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#events
  stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
    connected = state === ConnectionState.CONNECTED;

    if (connected) {
      joining = false;
      controls.classList.remove("hidden");
    } else {
      controls.classList.add("hidden");
    }
  });

  stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
    console.log("Participant Joined:", participant);
  });

  stage.on(StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED, (participant, streams) => {
    console.log("Participant Media Added: ", participant, streams);

    let streamsToDisplay = streams;

    if (participant.isLocal) {
      // Ensure to exclude local audio streams, otherwise echo will occur
      streamsToDisplay = streams.filter((stream) => stream.streamType ===
StreamType.VIDEO);
```

```
    }

    const videoEl = setupParticipant(participant);
    streamsToDisplay.forEach((stream) =>
 videoEl.srcObject.addTrack(stream.mediaStreamTrack));
  });

  stage.on(StageEvents.STAGE_PARTICIPANT_LEFT, (participant) => {
    console.log("Participant Left: ", participant);
    teardownParticipant(participant);
  });

  try {
    await stage.join();
  } catch (err) {
    joining = false;
    connected = false;
    console.error(err.message);
  }
};

const leaveStage = async () => {
  stage.leave();

  joining = false;
  connected = false;

  cameraButton.innerText = "Hide Camera";
  micButton.innerText = "Mute Mic";
  controls.classList.add("hidden");
};

function replaceBackground(result) {
  let imageData = canvasCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  let backgroundData = backgroundCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  const mask = result.categoryMask.getAsFloat32Array();
  let j = 0;

  for (let i = 0; i < mask.length; ++i) {
    const maskVal = Math.round(mask[i] * 255.0);

    j += 4;
```

```
    if (maskVal < 255) {
      backgroundData[j] = imageData[j];
      backgroundData[j + 1] = imageData[j + 1];
      backgroundData[j + 2] = imageData[j + 2];
      backgroundData[j + 3] = imageData[j + 3];
    }
  }
  const uint8Array = new Uint8ClampedArray(backgroundData.buffer);
  const dataNew = new ImageData(uint8Array, video.videoWidth, video.videoHeight);
  canvasCtx.putImageData(dataNew, 0, 0);
  window.requestAnimationFrame(renderVideoToCanvas);
}

const createImageSegmenter = async () => {
  const audio = await FilesetResolver.forVisionTasks("https://cdn.jsdelivr.net/npm/
@mediapipe/tasks-vision@0.10.2/wasm");

  imageSegmenter = await ImageSegmenter.createFromOptions(audio, {
    baseOptions: {
      modelAssetPath: "https://storage.googleapis.com/mediapipe-models/image_segmenter/
selfie_segmenter/float16/latest/selfie_segmenter.tflite",
      delegate: "GPU",
    },
    runningMode: "VIDEO",
    outputCategoryMask: true,
  });
};

const renderVideoToCanvas = async () => {
  if (video.currentTime === lastWebcamTime) {
    window.requestAnimationFrame(renderVideoToCanvas);
    return;
  }
  lastWebcamTime = video.currentTime;
  canvasCtx.drawImage(video, 0, 0, video.videoWidth, video.videoHeight);

  if (imageSegmenter === undefined) {
    return;
  }

  let startTimeMs = performance.now();

  imageSegmenter.segmentForVideo(video, startTimeMs, replaceBackground);
};
```

```
const initBackgroundCanvas = () => {
  let img = new Image();
  img.src = "beach.jpg";

  img.onload = () => {
    backgroundCtx.clearRect(0, 0, canvas.width, canvas.height);
    backgroundCtx.drawImage(img, 0, 0);
  };
};

createImageSegmenter();
init();
```

Cree un archivo de configuración de Webpack

Agregue esta configuración a su archivo de configuración de Webpack para empaquetar`app.js`, de modo que las llamadas de importación funcionen:

JavaScript

```
const path = require("path");
module.exports = {
  entry: ["./app.js"],
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist"),
  },
};
```

Agrupe sus archivos JavaScript

```
npm run build
```

Inicie un servidor HTTP simple desde el directorio que lo contiene `index.html` y `localhost:8000` ábralo para ver el resultado:

```
python3 -m http.server -d ./
```

Android

Para reemplazar el fondo de tu transmisión en directo, puedes usar la API de segmentación de selfies del Google ML Kit. La API de segmentación de selfies acepta una imagen de cámara como entrada y devuelve una máscara que proporciona una puntuación de confianza para cada píxel de la imagen, que indica si estaba en primer plano o en segundo plano. En función de la puntuación de confianza, puedes recuperar el color de píxel correspondiente de la imagen de fondo o de la imagen de primer plano. Este proceso continúa hasta que se hayan examinado todas las puntuaciones de confianza de la máscara. El resultado es una nueva matriz de colores de píxeles que contiene los píxeles del primer plano combinados con los píxeles de la imagen de fondo.

Para integrar la sustitución en segundo plano de con el SDK de transmisión para streaming en tiempo real de IVS, debe:

1. Instala las bibliotecas CameraX y el kit ML de Google.
2. Inicialice las variables repetitivas.
3. Creación de una fuente de imágenes personalizada.
4. Administra los fotogramas de las cámaras.
5. Transfiere los marcos de las cámaras al Google ML Kit.
6. Superponga el primer plano del marco de la cámara sobre su fondo personalizado.
7. Introduce la nueva imagen en una fuente de imágenes personalizada.

Instale las bibliotecas CameraX y el kit ML de Google

Para extraer imágenes de la transmisión de la cámara en vivo, usa la biblioteca CameraX de Android. Para instalar la biblioteca CameraX y el kit ML de Google, añade lo siguiente al archivo del `build.gradle` módulo. Sustituya `${camerax_version}` y `${google_ml_kit_version}` por la última versión de las bibliotecas CameraX y Google ML Kit, respectivamente.

Java

```
implementation "com.google.mlkit:segmentation-selfie:${google_ml_kit_version}"
implementation "androidx.camera:camera-core:${camerax_version}"
implementation "androidx.camera:camera-lifecycle:${camerax_version}"
```

Importe las siguientes bibliotecas:

Java

```
import androidx.camera.core.CameraSelector
import androidx.camera.core.ImageAnalysis
import androidx.camera.core.ImageProxy
import androidx.camera.lifecycle.ProcessCameraProvider
import com.google.mlkit.vision.segmentation.selfie.SelfieSegmenterOptions
```

Inicialización de variables reutilizable

Inicializa una instancia `ImageAnalysis` y una instancia de: `ExecutorService`

Java

```
private lateinit var binding: ActivityMainBinding
private lateinit var cameraExecutor: ExecutorService
private var analysisUseCase: ImageAnalysis? = null
```

Inicializa una instancia de Segmenter en STREAM_MODE:

Java

```
private val options =
        SelfieSegmenterOptions.Builder()
            .setDetectorMode(SelfieSegmenterOptions.STREAM_MODE)
            .build()

private val segmenter = Segmentation.getClient(options)
```

Creación de una fuente de imágenes personalizada

En el on`Create` método de tu actividad, crea una instancia de un `DeviceDiscovery` objeto y crea una fuente de imágenes personalizada. La imagen `Surface` proporcionada por la fuente de imagen personalizada recibirá la imagen final, con el primer plano superpuesto sobre una imagen de fondo personalizada. A continuación, creará una instancia de a `ImageLocalStageStream` utilizando la fuente de imagen personalizada. Luego, la instancia de a `ImageLocalStageStream` (nombrada `filterStream` en este ejemplo) se puede publicar en un escenario. Consulta la guía del SDK de IVS para Android Broadcast para obtener instrucciones sobre cómo configurar un escenario. Por último, cree también un hilo que se utilizará para gestionar la cámara.

Java

```
var deviceDiscovery = DeviceDiscovery(applicationContext)
var customSource = deviceDiscovery.createImageInputSource( BroadcastConfiguration.Vec2(
720F, 1280F
))
var surface: Surface = customSource.inputSurface
var filterStream = ImageLocalStageStream(customSource)

cameraExecutor = Executors.newSingleThreadExecutor()
```

Administrar tramas de cámara

A continuación, cree una función para inicializar la cámara. Esta función utiliza la biblioteca CameraX para extraer imágenes de la transmisión de la cámara en directo. En primer lugar, se crea una instancia de una `ProcessCameraProvider` llamada`cameraProviderFuture`. Este objeto representa un resultado futuro de la obtención de un proveedor de cámaras. A continuación, carga una imagen del proyecto en forma de mapa de bits. En este ejemplo se utiliza la imagen de una playa como fondo, pero puede ser cualquier imagen que desees.

A continuación, añada un oyente `acameraProviderFuture`. Este oyente recibe una notificación cuando la cámara está disponible o si se produce un error durante el proceso de obtención de un proveedor de cámaras.

Java

```
private fun startCamera(surface: Surface) {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
        val imageResource = R.drawable.beach
        val bgBitmap: Bitmap = BitmapFactory.decodeResource(resources, imageResource)
        var resultBitmap: Bitmap;


        cameraProviderFuture.addListener({
            val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()


                if (mediaImage != null) {
                    val inputImage =
                        InputImage.fromMediaImage(mediaImage,
 imageProxy.imageInfo.rotationDegrees)
```

```
                            resultBitmap = overlayForeground(mask, maskWidth,
maskHeight, inputBitmap, backgroundPixels)
                            canvas = surface.lockCanvas(null);
                            canvas.drawBitmap(resultBitmap, 0f, 0f, null)

                            surface.unlockCanvasAndPost(canvas);

                        }
                        .addOnFailureListener { exception ->
                            Log.d("App", exception.message!!)
                        }
                        .addOnCompleteListener {
                            imageProxy.close()
                        }

                }
            };

            val cameraSelector = CameraSelector.DEFAULT_FRONT_CAMERA

            try {
                // Unbind use cases before rebinding
                cameraProvider.unbindAll()

                // Bind use cases to camera
                cameraProvider.bindToLifecycle(this, cameraSelector, analysisUseCase)

            } catch(exc: Exception) {
                Log.e(TAG, "Use case binding failed", exc)
            }

        }, ContextCompat.getMainExecutor(this))
    }
```

En el oyente, cree `ImageAnalysis.Builder` para acceder a cada fotograma individual de la transmisión de la cámara en directo. Establezca la estrategia de contrapresión en. `STRATEGY_KEEP_ONLY_LATEST` Esto garantiza que solo se entregue un cuadro de cámara a la vez para su procesamiento. Convierte cada fotograma de cámara individual en un mapa de bits, de forma que puedas extraer sus píxeles y luego combinarlos con la imagen de fondo personalizada.

Java

```
val imageAnalyzer = ImageAnalysis.Builder()
```

```
analysisUseCase = imageAnalyzer
    .setTargetResolution(Size(360, 640))
    .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
    .build()

analysisUseCase?.setAnalyzer(cameraExecutor) { imageProxy: ImageProxy ->
    val mediaImage = imageProxy.image
    val tempBitmap = imageProxy.toBitmap();
    val inputBitmap = tempBitmap.rotate(imageProxy.imageInfo.rotationDegrees.toFloat())
```

Transfiere los marcos de cámara al kit ML de Google

A continuación, crea una `InputImage` y pásala a la instancia de Segmenter para su procesamiento. Se puede crear una `InputImage` a partir de una `ImageProxy` proporcionada por la instancia de `ImageAnalysis`. Una vez que `InputImage` se proporciona a Segmenter, devuelve una máscara con puntuaciones de confianza que indican la probabilidad de que un píxel esté en primer plano o en segundo plano. Esta máscara también proporciona propiedades de ancho y alto, que utilizará para crear una nueva matriz que contenga los píxeles de fondo de la imagen de fondo personalizada cargada anteriormente.

Java

```
if (mediaImage != null) {
        val inputImage =
            InputImage.fromMediaImag


segmenter.process(inputImage)
    .addOnSuccessListener { segmentationMask ->
        val mask = segmentationMask.buffer
        val maskWidth = segmentationMask.width
        val maskHeight = segmentationMask.height
        val backgroundPixels = IntArray(maskWidth * maskHeight)
        bgBitmap.getPixels(backgroundPixels, 0, maskWidth, 0, 0, maskWidth, maskHeight)
```

Superponga el primer plano del marco de la cámara sobre el fondo personalizado

Con la máscara que contiene las puntuaciones de confianza, el marco de la cámara como mapa de bits y los píxeles de color de la imagen de fondo personalizada, tienes todo lo que necesitas para superponer el primer plano al fondo personalizado. A continuación, se invoca a la `overlayForeground` función de con los siguientes parámetros:

Java

```
resultBitmap = overlayForeground(mask, maskWidth, maskHeight, inputBitmap,
 backgroundPixels)
```

Esta función recorre la máscara y comprueba los valores de confianza para determinar si se debe obtener el color de píxel correspondiente de la imagen de fondo o del marco de la cámara. Si el valor de confianza indica que lo más probable es que un píxel de la máscara esté en segundo plano, obtendrá el color de píxel correspondiente de la imagen de fondo; de lo contrario, obtendrá el color de píxel correspondiente del marco de la cámara para crear el primer plano. Una vez que la función termine de recorrer la máscara, se crea un nuevo mapa de bits con la nueva matriz de píxeles de color y se devuelve. Este nuevo mapa de bits contiene el primer plano superpuesto sobre el fondo personalizado.

Java

```
private fun overlayForeground(
        byteBuffer: ByteBuffer,
        maskWidth: Int,
        maskHeight: Int,
        cameraBitmap: Bitmap,
        backgroundPixels: IntArray
    ): Bitmap {
        @ColorInt val colors = IntArray(maskWidth * maskHeight)
        val cameraPixels = IntArray(maskWidth * maskHeight)

        cameraBitmap.getPixels(cameraPixels, 0, maskWidth, 0, 0, maskWidth, maskHeight)

        for (i in 0 until maskWidth * maskHeight) {
            val backgroundLikelihood: Float = 1 - byteBuffer.getFloat()

            // Apply the virtual background to the color if it's not part of the
 foreground
            if (backgroundLikelihood > 0.9) {
                // Get the corresponding pixel color from the background image
                // Set the color in the mask based on the background image pixel color
                colors[i] = backgroundPixels.get(i)
            } else {
                // Get the corresponding pixel color from the camera frame
                // Set the color in the mask based on the camera image pixel color
                colors[i] = cameraPixels.get(i)
            }
```

```
        }

        return Bitmap.createBitmap(
            colors, maskWidth, maskHeight, Bitmap.Config.ARGB_8888
        )
    }
```

Introduce la nueva imagen en una fuente de imagen personalizada

A continuación, puede escribir el nuevo mapa de bits en el `Surface` proporcionado por una fuente de imagen personalizada. Esto lo transmitirá a tu escenario.

Java

```
resultBitmap = overlayForeground(mask, inputBitmap, mutableBitmap, bgBitmap)
canvas = surface.lockCanvas(null);
canvas.drawBitmap(resultBitmap, 0f, 0f, null)
```

Esta es la función completa para obtener los fotogramas de la cámara, pasarlos a Segmenter y superponerlos sobre el fondo:

Java

```
@androidx.annotation.OptIn(androidx.camera.core.ExperimentalGetImage::class)
    private fun startCamera(surface: Surface) {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
        val imageResource = R.drawable.clouds
        val bgBitmap: Bitmap = BitmapFactory.decodeResource(resources, imageResource)
        var resultBitmap: Bitmap;

        cameraProviderFuture.addListener({
            // Used to bind the lifecycle of cameras to the lifecycle owner
            val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

            val imageAnalyzer = ImageAnalysis.Builder()
            analysisUseCase = imageAnalyzer
                .setTargetResolution(Size(720, 1280))
                .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                .build()

            analysisUseCase!!.setAnalyzer(cameraExecutor) { imageProxy: ImageProxy ->
                val mediaImage = imageProxy.image
                val tempBitmap = imageProxy.toBitmap();
```

```
                val inputBitmap =
tempBitmap.rotate(imageProxy.imageInfo.rotationDegrees.toFloat())

                if (mediaImage != null) {
                    val inputImage =
                        InputImage.fromMediaImage(mediaImage,
imageProxy.imageInfo.rotationDegrees)

                    segmenter.process(inputImage)
                        .addOnSuccessListener { segmentationMask ->
                            val mask = segmentationMask.buffer
                            val maskWidth = segmentationMask.width
                            val maskHeight = segmentationMask.height
                            val backgroundPixels = IntArray(maskWidth * maskHeight)
                            bgBitmap.getPixels(backgroundPixels, 0, maskWidth, 0, 0,
maskWidth, maskHeight)

                            resultBitmap = overlayForeground(mask, maskWidth,
maskHeight, inputBitmap, backgroundPixels)
                            canvas = surface.lockCanvas(null);
                            canvas.drawBitmap(resultBitmap, 0f, 0f, null)

                            surface.unlockCanvasAndPost(canvas);

                        }
                        .addOnFailureListener { exception ->
                            Log.d("App", exception.message!!)
                        }
                        .addOnCompleteListener {
                            imageProxy.close()
                        }

                }
            };

            val cameraSelector = CameraSelector.DEFAULT_FRONT_CAMERA

            try {
                // Unbind use cases before rebinding
                cameraProvider.unbindAll()

                // Bind use cases to camera
                cameraProvider.bindToLifecycle(this, cameraSelector, analysisUseCase)
```

```
            } catch(exc: Exception) {
                Log.e(TAG, "Use case binding failed", exc)
            }

        }, ContextCompat.getMainExecutor(this))
    }
```

SDK de transmisión de IVS: modos de audio móvil (transmisión en tiempo real)

La calidad del audio es una parte importante de la experiencia multimedia de cualquier equipo real, y no existe una configuración de audio única que funcione mejor para todos los casos de uso. Para garantizar que sus usuarios disfruten de la mejor experiencia al escuchar una transmisión en tiempo real de IVS, nuestros SDK para dispositivos móviles ofrecen varias configuraciones de audio predefinidas, así como personalizaciones más potentes, según sea necesario.

Introducción

Los SDK de transmisión móvil de IVS ofrecen una clase `StageAudioManager`. Esta clase está diseñada para ser el único punto de contacto para controlar los modos de audio subyacentes en ambas plataformas. En Android, esto controla el AudioManager, incluidos el modo de audio, la fuente de audio, el tipo de contenido, el uso y los dispositivos de comunicación. En iOS, controla la aplicación AVAudioSession, así como si VoiceProcessing está habilitado.

Importante: no interactúe con `AVAudioSession` ni `AudioManager` directamente mientras el SDK de transmisión en tiempo real de IVS esté activo. Si lo hace, podría perderse el audio o grabarse o reproducirse en el dispositivo incorrecto.

Antes de crear el primer objeto `DeviceDiscovery` o `Stage`, debe estar configurada la clase `StageAudioManager`.

Android (Kotlin)

```
StageAudioManager.getInstance(context).setPreset(StageAudioManager.UseCasePreset.VIDEO_CHAT)
 The default value

val deviceDiscovery = DeviceDiscovery(context)
val stage = Stage(context, token, this)

// Other Stage implementation code
```

iOS (Swift)

```
IVSStageAudioManager.sharedInstance().setPreset(.videoChat) // The default value

let deviceDiscovery = IVSDeviceDiscovery()
let stage = try? IVSStage(token: token, strategy: self)

// Other Stage implementation code
```

Si no hay nada establecido en `StageAudioManager` antes de la inicialización de una instancia `DeviceDiscovery` o `Stage`, el ajuste preestablecido `VideoChat` se aplica automáticamente.

Modo preestablecido de audio

El SDK de transmisión en tiempo real proporciona tres ajustes preestablecidos, cada uno adaptado a los casos de uso más comunes, tal y como se describe a continuación. Para cada ajuste preestablecido, cubrimos cinco categorías clave que diferencian los ajustes preestablecidos entre sí.

Chat de vídeo

Este es el ajuste preestablecido predeterminado, diseñado para cuando el dispositivo local va a mantener una conversación en tiempo real con otros participantes.

Categoría	Android	iOS
Cancelación de eco	Habilitado	Habilitado
Control de volumen	Volumen de llamadas	Volumen de llamadas
Selección de micrófono	Limitado según el sistema operativo. Es posible que los micrófonos USB no estén disponibl es.	Limitado según el sistema operativo. Es posible que los micrófonos USB y Bluetooth no estén disponibles. Deberían funcionar los auricular es Bluetooth que admiten tanto la

Categoría	Android	iOS
		entrada como la salida al mismo tiempo; por ejemplo, los AirPods.
Salida de audio	Cualquier dispositivo de salida debería funcionar.	Limitado según el sistema operativo. Es posible que los auriculares con cable no estén disponibles.
Calidad de audio	Media/baja. Sonará como una llamada telefónica, no como una reproducción multimedia.	Media/baja. Sonará como una llamada telefónica, no como una reproducción multimedia.

Suscripción única

Este ajuste preestablecido está diseñado para suscribir a otros participantes de la publicación, pero no para publicar usted mismo. Se centra en la calidad del audio y es compatible con todos los dispositivos de salida disponibles.

Categoría	Android	iOS
Cancelación de eco	Deshabilitad	Deshabilitad
Control de volumen	Volumen multimedia	Volumen multimedia
Selección de micrófono	N/A, este ajuste preestablecido no está diseñado para su publicación.	N/A, este ajuste preestablecido no está diseñado para su publicación.
Salida de audio	Cualquier dispositivo de salida debería funcionar.	Cualquier dispositivo de salida debería funcionar.
Calidad de audio	Alta. Cualquier tipo multimedia debería mostrarse con claridad, incluida la música.	Alta. Cualquier tipo multimedia debería mostrarse con claridad, incluida la música.

Studio

Este ajuste preestablecido está diseñado para una suscripción de alta calidad y, al mismo tiempo, permite publicar. Requiere el hardware de grabación y reproducción para poder cancelar el eco. Un caso de uso en este caso sería usar un micrófono USB y unos auriculares con cable. El SDK mantendrá la más alta calidad de audio y, al mismo tiempo, se basará en la separación física de esos dispositivos para evitar que produzcan eco.

Categoría	Android	iOS
Cancelación de eco	Deshabilitad	Deshabilitad
Control de volumen	Volumen multimedia en la mayoría de los casos. Volumen de llamadas cuando hay un micrófono Bluetooth conectado.	Volumen multimedia
Selección de micrófono	Cualquier micrófono debería funcionar.	Cualquier micrófono debería funcionar.
Salida de audio	Cualquier dispositivo de salida debería funcionar.	Cualquier dispositivo de salida debería funcionar.
Calidad de audio	Alta. Ambos lados deberían poder enviar música y escucharla con claridad en el otro lado. Cuando se conecta un auricular Bluetooth, la calidad del audio disminuirá debido a que el modo Bluetooth SCO está activado.	Alta. Ambos lados deberían poder enviar música y escucharla con claridad en el otro lado. Cuando se conecta un auricular Bluetooth, la calidad del audio puede disminuir debido a la activación del modo Bluetooth SCO, en función del auricular.

Casos de uso avanzados

Más allá de los ajustes preestablecidos, los SDK de transmisión en tiempo real de iOS y Android permiten configurar los modos de audio de la plataforma subyacente:

- En Android, configure AudioSource, Usage y ContentType.
- En iOS, use AVAudioSession.Category, AVAudioSession.CategoryOptions, AVAudioSession.Mode y la posibilidad de activar o no el procesamiento de voz durante la publicación.

Android (Kotlin)

```
// This would act similar to the Subscribe Only preset, but it uses a different
 ContentType.
StageAudioManager.getInstance(context)
    .setConfiguration(StageAudioManager.Source.GENERIC,
                      StageAudioManager.ContentType.MOVIE,
                      StageAudioManager.Usage.MEDIA);

val stage = Stage(context, token, this)

// Other Stage implementation code
```

iOS (Swift)

```
// This would act similar to the Subscribe Only preset, but it uses a different mode
 and options.
IVSStageAudioManager.sharedInstance()
    .setCategory(.playback,
                 options: [.duckOthers, .mixWithOthers],
                 mode: .default)

let stage = try? IVSStage(token: token, strategy: self)

// Other Stage implementation code
```

Publicación con Bluetooth en Android

El SDK vuelve automáticamente al `VIDEO_CHAT` preestablecido en Android cuando se cumplen las siguientes condiciones:

- La configuración asignada no usa el valor de uso `VOICE_COMMUNICATION`.
- Hay un micrófono Bluetooth conectado al dispositivo.

- El participante local está publicando en un escenario.

Esta es una limitación del sistema operativo Android en lo que respecta al uso de auriculares Bluetooth para grabar audio.

Integración con otros servicios de SDK

Como tanto iOS como Android admiten solo un modo de audio activo por aplicación, es habitual que surjan conflictos si la aplicación utiliza varios SDK que requieren el control del modo de audio. Cuando se encuentre con estos conflictos, puede probar algunas estrategias de resolución habituales, que se explican a continuación.

Coincidencia de los valores del modo de audio

Con las opciones de configuración de audio avanzadas del SDK de IVS o las funciones de otro SDK, haga que los dos SDK se alineen con los valores subyacentes.

Ágora

iOS

En iOS, decirle al SDK de Agora que mantenga la `AVAudioSession` activa evitará que se desactive mientras el SDK de transmisión en tiempo real de IVS lo esté utilizando.

```
myRtcEngine.SetParameters("{\"che.audio.keep.audiosession\":true}");
```

Android

Evita llamar `setEnableSpeakerphone` en `RtcEngine`, y llame a `enableLocalAudio(false)` mientras publica con el SDK de transmisión en tiempo real de IVS. Puede volver a llamar a `enableLocalAudio(true)` cuando el SDK de IVS no esté publicándose.

Uso de Amazon EventBridge con la transmisión en tiempo real de IVS

Puede utilizar Amazon EventBridge para monitorear sus transmisiones de Amazon Interactive Video Service (IVS).

Amazon IVS envía eventos de cambio sobre el estado de sus transmisiones a Amazon EventBridge. Todos los eventos que se entregan son válidos. Sin embargo, los eventos se envían en la medida de lo posible, lo que significa que no hay garantía de que:

- Los eventos se entregan: puede producirse un evento designado (por ejemplo, se publica un participante), pero es posible que Amazon IVS no envíe un evento correspondiente a EventBridge. Amazon IVS intenta entregar eventos durante varias horas antes de darse por vencido.
- Los eventos que se entregan llegarán en un periodo de tiempo especificado: es posible que reciba eventos de hasta unas pocas horas de antigüedad.
- Los eventos se entregan en orden: los eventos pueden estar desordenados, especialmente si se envían con poco tiempo de diferencia. Por ejemplo, puede ver un participante no publicado antes que un participante publicado.

Si bien es raro que los eventos falten, se retrasen o estén fuera de secuencia, debe gestionar estas posibilidades si escribe programas críticos para el negocio que dependen del orden o la existencia de los eventos de notificación.

Puede crear reglas de EventBridge para cualquiera de los siguientes eventos.

Tipo de evento	Evento	Enviado cuando…
Cambio de estado de composición del IVS	Error en el destino	Falló un intento de enviar la salida a un destino. Por ejemplo, se produjo un error en la transmisión a un canal porque no había ninguna clave de transmisión o se estaba produciendo otra transmisión.
Cambio de estado de composición del IVS	Inicio de destino	La salida a un destino se inició correctamente.

Tipo de evento	Evento	Enviado cuando…
Cambio de estado de composición del IVS	Fin de destino	La salida a un destino ha finalizado.
Cambio de estado de composición del IVS	Reconexión de destino	Se interrumpió la salida a un destino y se está intentando volver a conectar.
Cambio de estado de composición del IVS	Inicio de la sesión	Se creó una sesión de Composición. Este evento se desencadena cuando una canalización de un proceso de Composición se inicializa correctam ente. En este momento, el proceso de composición se ha suscrito correctamente a un Escenario y recibe contenido multimedia y puede componer vídeo.
Cambio de estado de composición del IVS	Fin de la sesión	Se ha completado una sesión de composición.
Cambio de estado de composición del IVS	Error de sesión	No se pudo inicializar una canalizac ión de Composición debido a que los recursos de Escenario no estaban disponibles o a algún otro error interno.
Actualización de una fase de IVS	Participante publicado	Un participante comienza a publicar en una fase.
Actualización de una fase de IVS	Participante no publicado	Un participante ha dejado de publicar en una fase.

Creación de reglas de Amazon EventBridge para Amazon IVS

Puede crear una regla que se active en función de un evento que emita Amazon IVS. Siga los pasos que se indican en Crear una regla en Amazon EventBridge en la Guía del usuario de Amazon EventBridge. Cuando seleccione un servicio, elija Interactive Video Service (IVS).

Ejemplos: cambio del estado de la composición

Error de destino: este evento se envía cuando se produce un error al intentar enviarlo a un destino. Por ejemplo, la transmisión a un canal falló porque no había ninguna clave de transmisión o se estaba produciendo otra transmisión.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination Failure",
     "stage_arn": "<stage-arn>",
     "id": "<Destination-id>",
     "reason": "eg. stream key invalid"
   }
}
```

Inicio de destino: este evento se envía cuando la salida a un destino se inicia correctamente.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
```

```
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination Start",
     "stage_arn": "<stage-arn>",
     "id": "<destination-id>",
   }
}
```

Final de destino: este evento se envía cuando finaliza la salida a un destino.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination End",
     "stage_arn": "<stage-arn>",
     "id": "<Destination-id>",
   }
}
```

Reconexión de destino: este evento se envía cuando se interrumpe la salida a un destino y se intenta volver a conectar.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
```

```
    ],
    "detail": {
      "event_name": "Destination Reconnecting",
      "stage_arn": "<stage-arn>",
      "id": "<Destination-id>",
    }
}
```

Inicio de sesión: este evento se envía cuando se crea una sesión de composición. Este evento se desencadena cuando una canalización de un proceso de composición se inicializa correctamente. En este momento, el proceso de composición se ha suscrito correctamente a un escenario y recibe contenido multimedia y puede componer vídeo.

```
{
    "version": "0",
    "id": "01234567-0123-0123-0123-012345678901",
    "detail-type": "IVS Composition State Change",
    "source": "aws.ivs",
    "account": "aws_account_id",
    "time": "2017-06-12T10:23:43Z",
    "region": "us-east-1",
    "resources": [
      "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
    ],
    "detail": {
      "event_name": "Session Start",
      "stage_arn": "<stage-arn>"
    }
}
```

Fin de sesión: este evento se envía cuando se completa una sesión de composición y se eliminan todos los recursos.

```
{
    "version": "0",
    "id": "01234567-0123-0123-0123-012345678901",
    "detail-type": "IVS Composition State Change",
    "source": "aws.ivs",
    "account": "aws_account_id",
    "time": "2017-06-12T10:23:43Z",
    "region": "us-east-1",
    "resources": [
```

```
      "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session End",
     "stage_arn": "<stage-arn>"
   }
}
```

Error de sesión: este evento se envía cuando una canalización de composición no se pudo inicializar debido a que los recursos del escenario no estaban disponibles, no había participantes en el escenario o a algún otro error interno.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session Failure",
     "stage_arn": "<stage-arn>",
     "reason": "eg. no participants in the stage"
   }
}
```

Ejemplos: actualización de una fase

Los eventos de actualización de fases incluyen un nombre de evento (que clasifica el evento) y metadatos sobre el evento. Los metadatos incluyen el ID del participante que activó el evento, los ID de fase y sesión asociados y el ID de usuario.

Participante publicado: este evento se envía cuando un participante comienza a publicar en una fase.

```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
```

```
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-1234567890",
        "event_name": "Participant Published",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f"
    }
}
```

Participante no publicado: este evento se envía cuando un participante ha dejado de publicar en una fase.

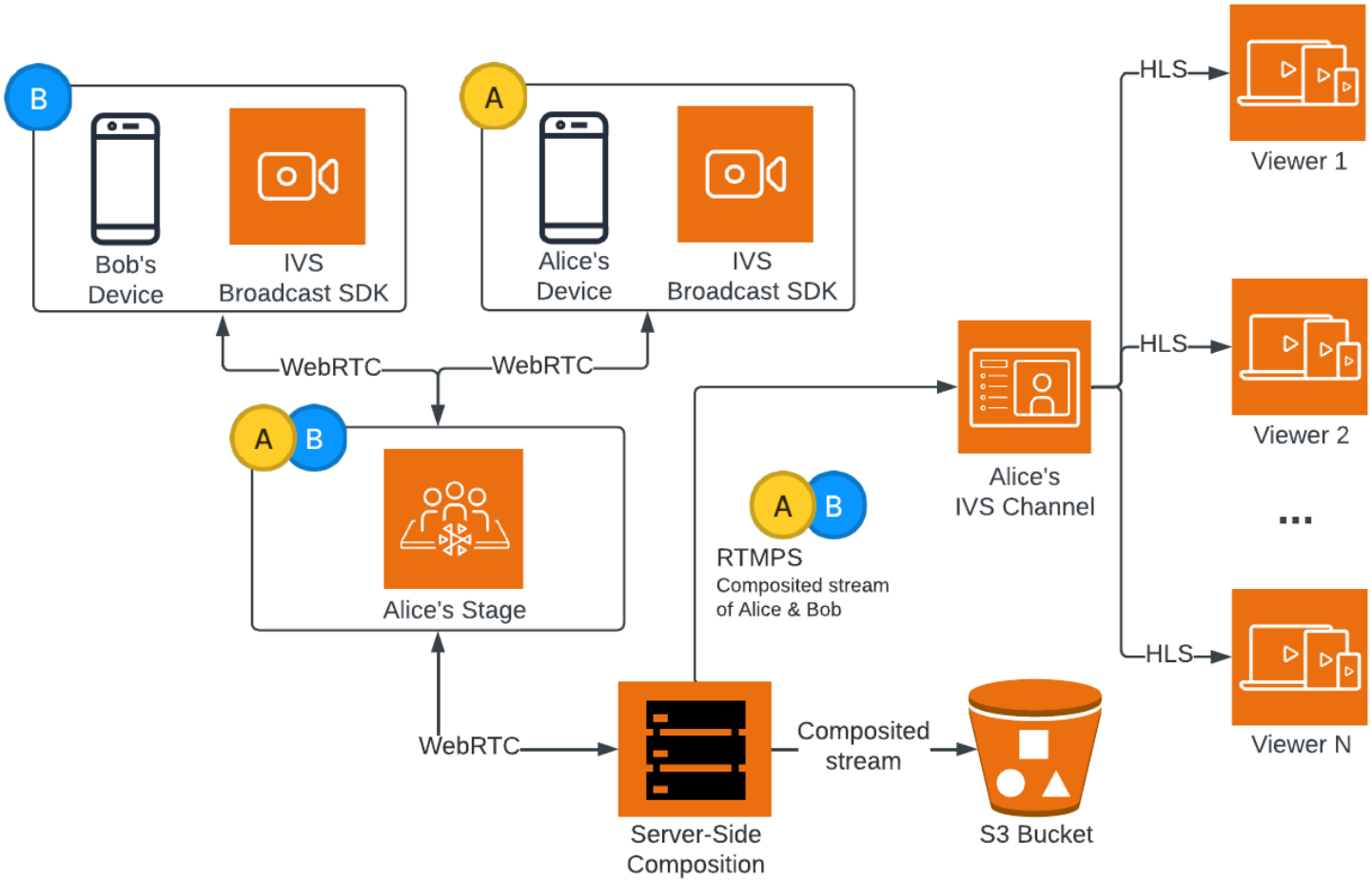
```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-1234567890",
        "event_name": "Participant Unpublished",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f"
    }
}
```

Composición del servidor (streaming en tiempo real)

La composición del servidor utiliza un servidor IVS para mezclar el audio y el vídeo de todos los participantes del escenario y, a continuación, envía este vídeo mixto a un canal IVS (por ejemplo, para llegar a un público más amplio) o a un bucket S3. La composición del servidor se invoca a través de los puntos de conexión del plano de control del IVS en la región de origen del escenario.

Transmitir o grabar un escenario utilizando la composición del servidor ofrece numerosas ventajas, lo que lo convierte en una opción atractiva para los usuarios que buscan flujos de trabajo de vídeo eficientes y fiables basados en la nube.

Este diagrama ilustra cómo funciona la composición del servidor:



Ventajas

En comparación con la composición del cliente, la composición del servidor tiene las siguientes ventajas:

- Reducción de la carga de clientes: con la composición del servidor, la carga de procesar y combinar las fuentes de audio y vídeo pasa de los dispositivos cliente individuales al propio servidor. La composición del servidor elimina la necesidad de que los dispositivos cliente utilicen sus recursos de CPU y red para componer la vista y transmitirla al IVS. Esto significa que los espectadores pueden ver la transmisión sin que sus dispositivos tengan que realizar tareas que consumen muchos recursos, lo que puede mejorar la duración de la batería y disfrutar de una experiencia de visualización más fluida.
- Calidad uniforme: la composición del servidor permite un control preciso de la calidad, la resolución y la velocidad de bits de la transmisión final. Esto garantiza una experiencia de visualización uniforme para todos los espectadores, independientemente de las capacidades de sus dispositivos individuales.
- Resiliencia: al centralizar el proceso de composición en el servidor, la transmisión se vuelve más sólida. Incluso si el dispositivo de un publicador experimenta limitaciones o fluctuaciones técnicas, el servidor puede adaptarse y ofrecer una transmisión más fluida a toda la audiencia.
- Eficiencia del ancho de banda: dado que el servidor se encarga de la composición, los publicadores de la presentación no tienen que gastar más ancho de banda para transmitir el vídeo a IVS.

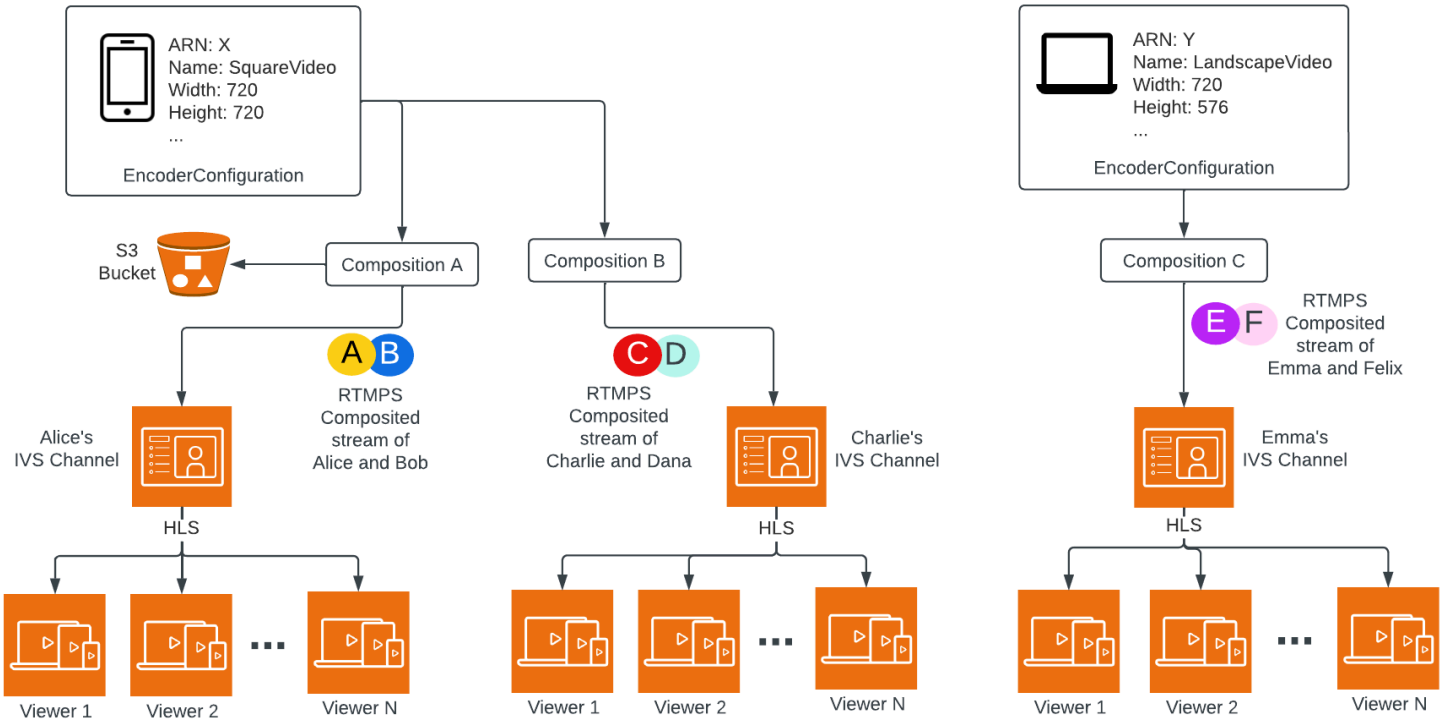
Como alternativa, para retransmitir una escena a un canal de IVS, puede realizar la composición desde el lado del cliente; consulte Habilitar varios hosts en una transmisión de IVS en la Guía del usuario de transmisión de baja latencia de IVS.

API de IVS

La composición del servidor utiliza estos elementos clave de la API:

- Un objeto EncoderConfiguration permite personalizar el formato del vídeo que se va a generar (altura, anchura, velocidad de bits y otros parámetros de transmisión). Puede reutilizar un EncoderConfiguration cada vez que llame al punto de conexión StartComposition.
- Los puntos de conexión de Composición rastrean la composición del vídeo y lo envían a un canal IVS.
- StorageConfiguration rastrea el bucket S3 en el que se graban las composiciones.

Para utilizar la composición del servidor, debe crear una EncoderConfiguration y adjuntarla al llamar al punto de conexión StartComposition. En este ejemplo, la configuración de SquareVideo EncoderConfiguration se utiliza en dos composiciones:



Para obtener información completa, consulte la Referencia de la API de streaming en tiempo real de IVS.

Layouts (Diseños)

De forma predeterminada, la característica de composición del servidor utiliza un diseño de cuadrícula para organizar a los participantes del escenario en espacios del mismo tamaño:

Este diseño ofrece a los clientes la opción de configurar e invocar un espacio destacado. El espacio destacado aparece en la pantalla principal, y debajo se muestra a los demás participantes en el espacio del mismo tamaño:

Nota: la resolución máxima admitida por un publicador de escenarios en la composición del servidor es de 1080p. Si un publicador envía un vídeo de más de 1080p, se representará como participante únicamente de audio.

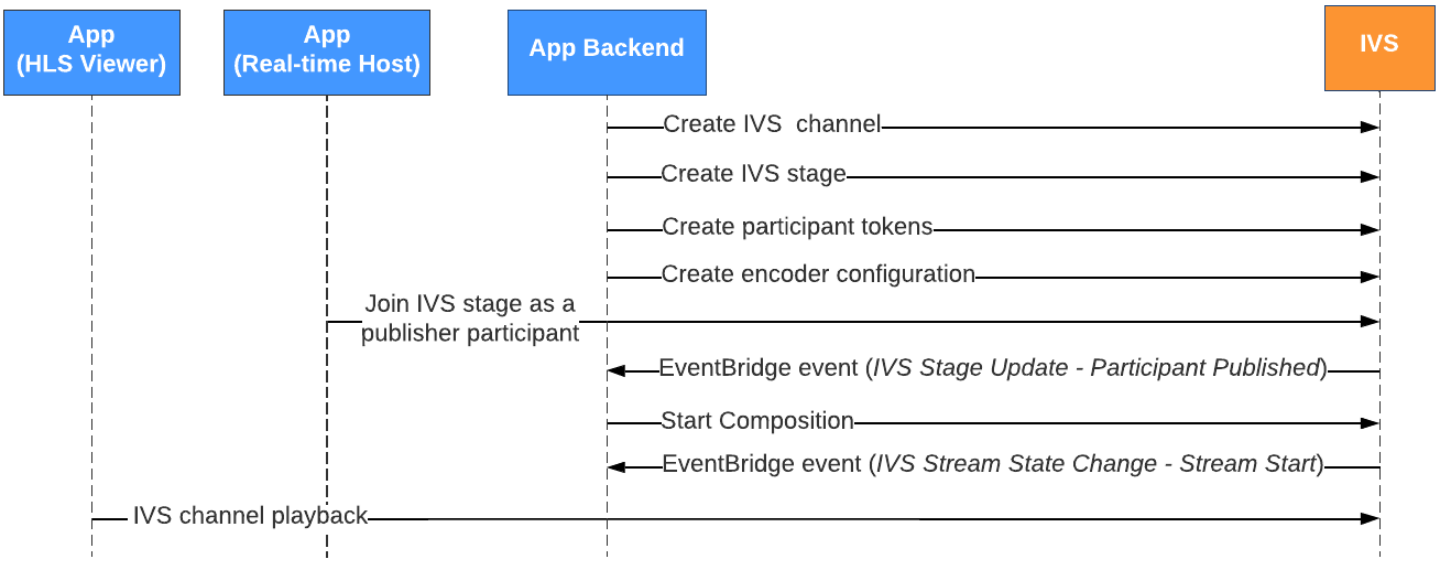
Introducción

Requisitos previos

Para utilizar la composición del servidor, debe tener un escenario con publicadores activos y utilizar un canal IVS o un bucket S3 como destino de la composición. A continuación, describimos un posible flujo de trabajo que utiliza los eventos de EventBridge para iniciar una composición que

retransmite el escenario a un canal de IVS cuando un participante publica. Como alternativa, puede iniciar y detener las composiciones según la lógica de su propia aplicación. Consulte Grabación del compuesto para ver otro ejemplo que muestra el uso de la composición del servidor para grabar un escenario directamente en un bucket S3.

1. Cree un canal IVS. Consulte Introducción a la transmisión de baja latencia de Amazon IVS.
2. Cree un escenario de IVS y tokens de participante para cada publicador.
3. Cree una configuración EncoderConfiguration.
4. Únase al escenario y publique en él. (Consulte las secciones "Publicación y suscripción" de las guías del SDK para transmisiones en tiempo real: Web, Android e iOS).
5. Cuando reciba un evento de EventBridge publicado por participantes, llame a StartComposition.
6. Espere unos segundos y vea la vista compuesta en la reproducción del canal.



Nota: una composición se apaga automáticamente después de 60 segundos de inactividad por parte de los publicadores participantes en el escenario. En ese momento, la composición finaliza y pasa a un estado `STOPPED`. Una composición se elimina automáticamente después de unos minutos en ese estado `STOPPED`.

Instrucciones de la CLI

El uso de AWS CLI es una opción avanzada y requiere que primero descargue y configure la CLI en su equipo. Para obtener más información, consulte la Guía del usuario de la interfaz de línea de comandos de AWS.

Ahora puede utilizar la CLI para crear y administrar recursos. Los puntos de conexión de la Composición se encuentran debajo del espacio de nombres `ivs-realtime`.

Cree el recurso de EncoderConfiguration

Un EncoderConfiguration es un objeto que permite personalizar el formato del vídeo generado (altura, anchura, velocidad de bits y otros parámetros de streaming). Puede reutilizar una EncoderConfiguration cada vez que llame al punto de conexión de Composición, como se explica en el siguiente paso.

El siguiente comando crea un recurso EncoderConfiguration que configura los parámetros de composición de vídeo del servidor, como la velocidad de bits del vídeo, la velocidad de fotogramas y la resolución:

```
aws ivs-realtime create-encoder-configuration --name "MyEncoderConfig" --video
 "bitrate=2500000,height=720,width=1280,framerate=30"
```

La respuesta es:

```
{
   "encoderConfiguration": {
      "arn": "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/9W59OBY2M8s4",
      "name": "MyEncoderConfig",
      "tags": {},
      "video": {
         "bitrate": 2500000,
         "framerate": 30,
         "height": 720,
         "width": 1280
      }
   }
}
```

Inicie una Composición

Con el ARN de EncoderConfiguration proporcionado en la respuesta anterior, cree su recurso de composición:

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
```

```
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/9W59OBY2M8s4"}}]'
```

La respuesta mostrará que la Composición se creó con un estado `STARTING`. Una vez que la Composición comience a publicarse, el estado pasará a `ACTIVE`. (Para ver el estado, llame al punto de conexión ListCompositions o GetComposition).

Una vez que una Composición sea `ACTIVE`, la vista compuesta del escenario IVS es visible en el canal IVS mediante ListCompositions:

```
aws ivs-realtime list-compositions
```

La respuesta es:

```
{
   "compositions": [
      {
         "arn": "arn:aws:ivs:us-east-1:927810967299:composition/YVoaXkKdEdRP",
         "destinations": [
            {
               "id": "bD9rRoN91fHU",
               "startTime": "2023-09-21T15:38:39+00:00",
               "state": "ACTIVE"
            }
         ],
         "stageArn": "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
         "startTime": "2023-09-21T15:38:37+00:00",
         "state": "ACTIVE",
         "tags": {}
      }
   ]
}
```

Nota: es necesario que los publicadores participantes publiquen activamente en el escenario para mantener viva la composición. Para obtener más información, consulte las secciones "Publicaciones y suscripciones” de las guías del SDK para transmisiones en tiempo real: Web, Android e iOS. Debe crear un token de escenario distinto para cada participante.

Habilitar pantalla compartida

Para usar un diseño de pantalla compartida fijo, siga los pasos que se indican a continuación.

Cree el recurso de EncoderConfiguration

El siguiente comando crea un recurso EncoderConfiguration que configura los parámetros de composición del servidor (velocidad de bits, velocidad de fotogramas y resolución del vídeo).

```
aws ivs-realtime create-encoder-configuration --name "test-ssc-with-screen-share" --
video={bitrate=2000000,framerate=30,height=720,width=1280}
```

Cree un token de participante en el escenario con un atributo `screen-share`. Como especificaremos `screen-share` como el nombre del espacio `featured`, necesitamos crear un token de escenario con el atributo `screen-share` establecido en `true`:

```
aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes screen-share=true
```

La respuesta es:

```
{
   "participantToken": {
      "attributes": {
         "screen-share": "true"
      },
      "expirationTime": "2023-08-04T05:26:11+00:00",
      "participantId": "E813MFklPWLF",
      "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2OTExMjY3NzEsImlhdCI6MTY5MTA4MzU3MSwianRpIjoiRT
   }
}
```

Comience la composición

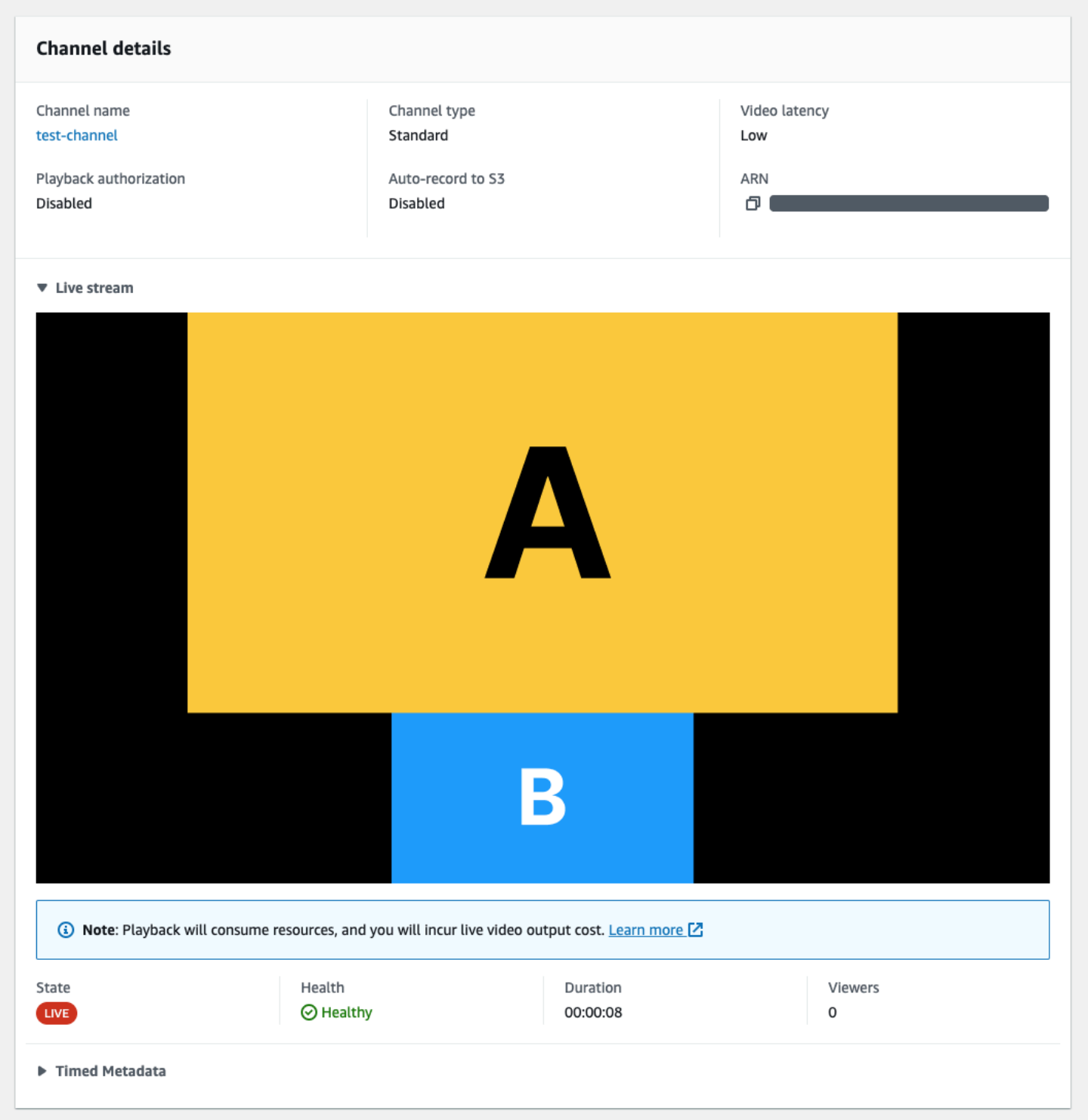
Para iniciar la composición con la característica de compartir pantalla, utilizamos este comando:

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVaOwO"}}]' --layout
 "grid={featuredParticipantAttribute=screen-share}"
```

La respuesta es:

```
{
   "composition" : {
      "arn" : "arn:aws:ivs:us-east-1:927810967299:composition/B19tQcXRgtoz",
      "destinations" : [ {
         "configuration" : {
            "channel" : {
               "channelArn" : "arn:aws:ivs:us-east-1:927810967299:channel/
DOlMW4dfMR8r",
               "encoderConfigurationArn" : "arn:aws:ivs:us-east-1:927810967299:encoder-
configuration/DEkQHWPVaOwO"
            },
            "name" : ""
         },
         "id" : "SGmgBXTULuXv",
         "state" : "STARTING"
      } ],
      "layout" : {
         "grid" : {
            "featuredParticipantAttribute" : "screen-share"
         }
      },
      "stageArn" : "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
      "startTime" : "2023-09-27T21:32:38Z",
      "state" : "STARTING",
      "tags" : { }
   }
}
```

Cuando el participante del escenario `E813MFklPWLF` se una al escenario, el vídeo de ese participante se mostrará en el espacio destacado y todos los demás publicadores del escenario se renderizarán debajo del espacio:



Detenga la composición

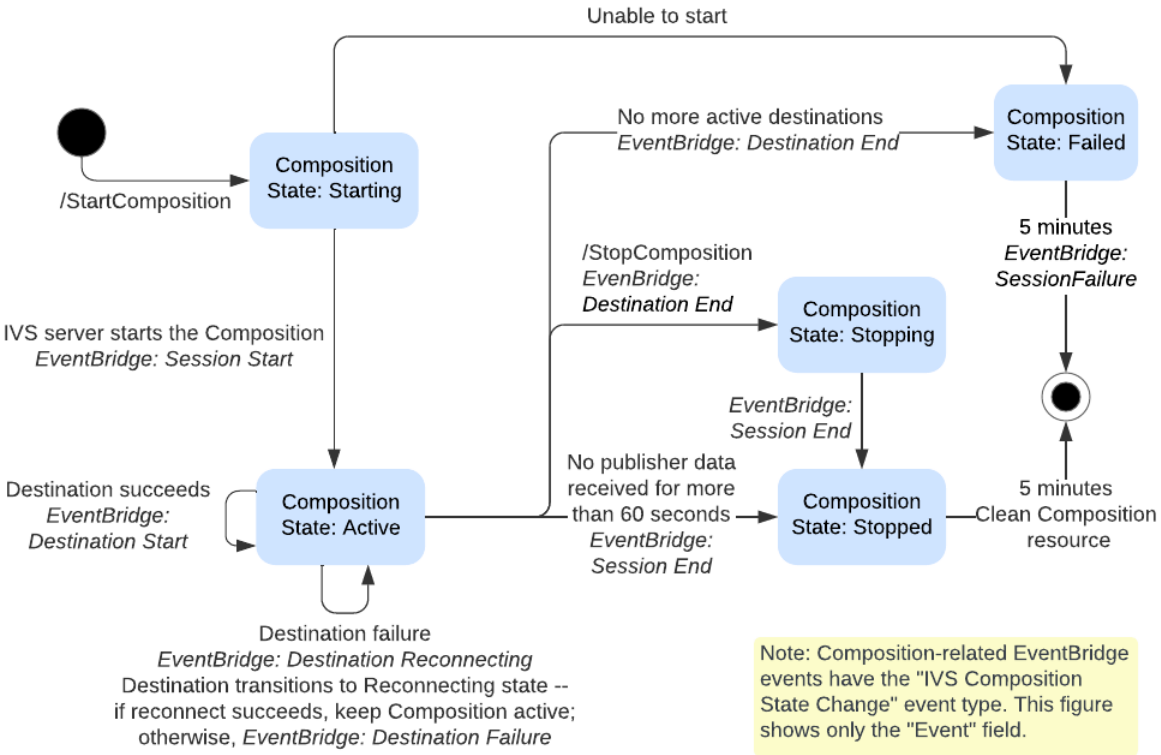
Para detener una composición en cualquier momento, llame al punto de conexión StopComposition:

```
aws ivs-realtime stop-composition --arn arn:aws:ivs:us-east-1:927810967299:composition/
B19tQcXRgtoz
```

Vida útil de la composición

Use el siguiente diagrama para entender las transiciones de estado de una Composición. A grandes rasgos, el ciclo de vida de una Composición es el siguiente:

1. Se crea un recurso de Composición cuando el usuario llama al punto de conexión StartComposition
2. Una vez que IVS inicie correctamente la composición, se enviará un evento EventBridge de “Cambio de estado de composición de IVS (inicio de sesión)”. Consulte Uso de EventBridge con el streaming en tiempo real de IVS para obtener más información sobre los eventos.
3. Una vez que una Composición está en estado activo, puede ocurrir lo siguiente:
   - El usuario detiene la Composición: si se llama al punto de conexión de StopComposition, IVS inicia un cierre correcto de la Composición y envía eventos de “Final de destino” seguidos de un evento de “Fin de sesión”.
   - La Composición se cierra automáticamente: si ningún participante publica activamente en la etapa IVS, la Composición se finaliza automáticamente después de 60 segundos y se envían los eventos de EventBridge.
   - Error en el destino: si un destino falla inesperadamente (por ejemplo, si se elimina el canal IVS), el destino pasa al estado `RECONNECTING` y se envía un evento de “Reconexión del destino”. Si la recuperación no es posible, el IVS cambia el destino al estado `FAILED` y se envía un evento de “Error en el destino”. El IVS mantiene viva la composición si al menos uno de sus destinos está activo.
4. Una vez que la composición está en el estado `STOPPED` o `FAILED`, se limpia automáticamente después de cinco minutos. (Entonces, ListCompositions o GetComposition ya no la recuperan).


Unable to start
/StartComposition
Composition State: Starting
No more active destinations
EventBridge: Destination End
Composition State: Failed
5 minutes
EventBridge: SessionFailure
/StopComposition
EvenBridge:
Destination End
Composition State: Stopping
IVS server starts the Composition
EventBridge: Session Start
EventBridge: Session End
Destination succeeds
EventBridge: Destination Start
Composition State: Active
No publisher data received for more than 60 seconds
EventBridge: Session End
Composition State: Stopped
5 minutes
Clean Composition resource
Destination failure
EventBridge: Destination Reconnecting
Destination transitions to Reconnecting state -- if reconnect succeeds, keep Composition active; otherwise, EventBridge: Destination Failure
Note: Composition-related EventBridge events have the "IVS Composition State Change" event type. This figure shows only the "Event" field.

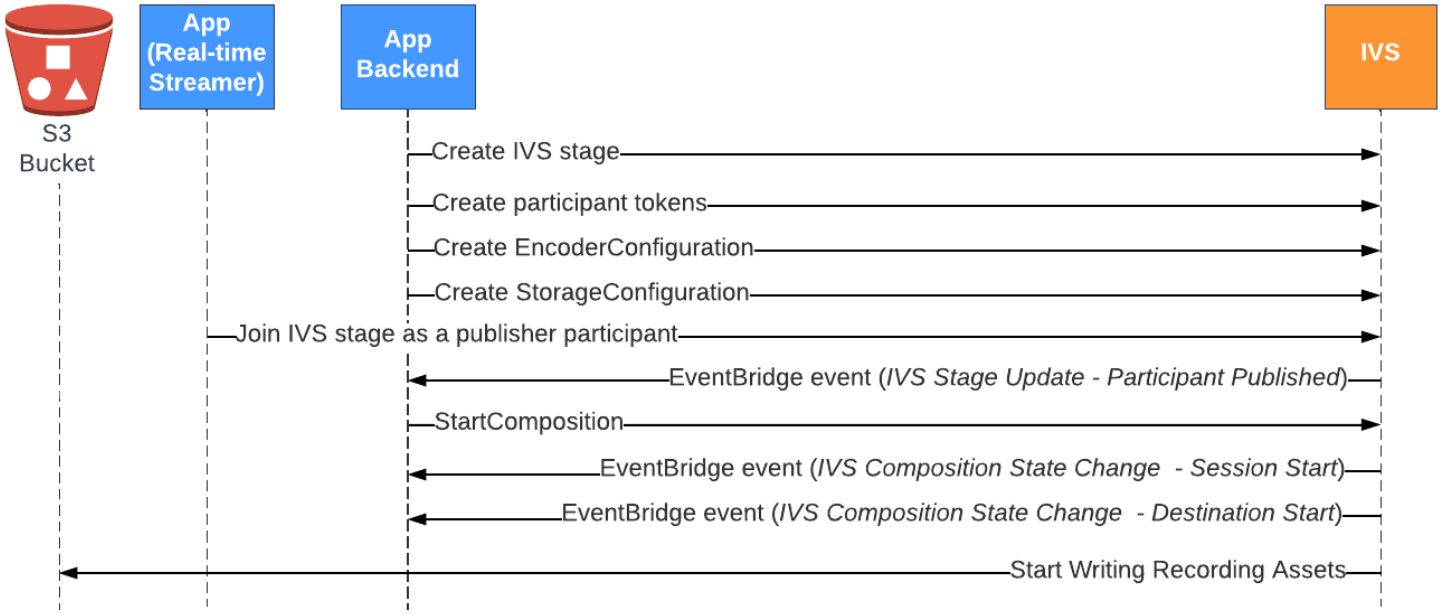
Grabación compuesta (transmisión en tiempo real)

Este documento explica cómo utilizar la característica de grabación compuesta en la composición del servidor. La grabación compuesta permite generar grabaciones HLS de un escenario IVS al combinar de forma eficaz todos los publicadores de escenarios en una sola vista mediante un servidor IVS y, a continuación, guardar el vídeo resultante en un bucket de S3.

Requisitos previos

Para utilizar la grabación compuesta, debe tener un escenario con editores activos y un bucket S3 para utilizarlo como destino de la grabación. A continuación, describimos un posible flujo de trabajo que utiliza EventBridge eventos para grabar una composición en un bucket de S3. Como alternativa, puede iniciar y detener las composiciones según la lógica de su propia aplicación.

1. Cree un escenario de IVS y tokens de participante para cada publicador.
2. Cree un EncoderConfiguration(un objeto que represente cómo debe renderizarse el vídeo grabado).
3. Cree un depósito S3 y un StorageConfiguration(donde se almacenará el contenido de la grabación).
4. Únase al escenario y publique en él.
5. Cuando reciba un EventBridge evento publicado por un participante, llame StartCompositioncon un DestinationConfiguration objeto S3 como destino
6. Tras unos segundos, debería poder ver cómo los segmentos HLS se conservan en sus buckets de S3.



Nota: Una composición se apaga automáticamente después de 60 segundos de inactividad por parte de los publicadores participantes en el escenario. En ese momento, la composición finaliza y pasa a un estado `STOPPED`. Una composición se borra automáticamente después de unos minutos en ese estado `STOPPED`. Para obtener más información, consulte Ciclo de vida de la composición en Composición del servidor.

Ejemplo de grabación compuesta: StartComposition con un destino de bucket S3

El siguiente ejemplo muestra una llamada típica al StartCompositionpunto final, especificando S3 como el único destino de la composición. Una vez que la composición pase a un estado `ACTIVE`, los segmentos de vídeo y los metadatos comenzarán a escribirse en el bucket de S3 especificado por el objeto `storageConfiguration`. Para crear composiciones con diferentes diseños, consulte la sección “Diseños” en Composición del servidor y la Referencia de la API de transmisión en tiempo real de IVS.

Solicitud

```
POST /StartComposition HTTP/1.1
Content-type: application/json

{
   "destinations": [
      {
```

```
         "s3": {
            "encoderConfigurationArns": [
              "arn:aws:ivs:ap-northeast-1:927810967299:encoder-configuration/
PAAwglkRtjge"
            ],
            "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq"
         }
      }
   ],
   "idempotencyToken": "db1i782f1g9",
   "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr"
}
```

Respuesta

```
{
    "composition": {
        "arn": "arn:aws:ivs:ap-northeast-1:927810967299:composition/s2AdaGUbvQgp",
        "destinations": [
            {
                "configuration": {
                    "name": "",
                    "s3": {
                        "encoderConfigurationArns": [
                            "arn:aws:ivs:ap-northeast-1:927810967299:encoder-
configuration/PAAwglkRtjge"
                        ],
                        "recordingConfiguration": {
                            "format": "HLS"
                        },
                        "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq"
                    }
                },
                "detail": {
                    "s3": {
                        "recordingPrefix": "MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/
composite"
                    }
                },
                "id": "2pBRKrNgX1ff",
                "state": "STARTING"
```

```
            }
        ],
        "layout": null,
        "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr",
        "startTime": "2023-11-01T06:25:37Z",
        "state": "STARTING",
        "tags": {}
    }
}
```

El `recordingPrefix` campo, presente en la StartComposition respuesta, se puede usar para determinar dónde se almacenará el contenido de la grabación.

Grabación de contenidos

Cuando la composición pase a un `ACTIVE` estado determinado, empezará a ver cómo se escriben segmentos de vídeo HLS y archivos de metadatos en el compartimento S3 que se proporcionó al realizar la llamada StartComposition. Estos contenidos están disponibles para el procesamiento posterior o la reproducción como video bajo demanda.

Tenga en cuenta que una vez que se publica una composición, se emite un evento de “Cambio en el estado de la composición de IVS” y se puede tardar un poco en escribir los archivos de manifiesto y los segmentos de video. Le recomendamos que reproduzca o procese transmisiones grabadas solo después de que se reciba el evento “Cambio en el estado de la composición de IVS (fin de la sesión)” Para obtener más información, consulte Uso EventBridge con IVS Real-Time Streaming.

A continuación, se muestra una estructura de directorios de ejemplo y el contenido de una grabación de una sesión en vivo de IVS:

```
MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/composite
   events
      recording-started.json
      recording-ended.json
   media
      hls
```

La carpeta `events` contiene los archivos de metadatos correspondientes al evento de grabación. Los archivos de metadatos JSON se generan cuando la grabación se inicia, finaliza correctamente o termina con errores:

- `events/recording-started.json`
- `events/recording-ended.json`
- `events/recording-failed.json`

Una carpeta `events` contendrá `recording-started.json` y `recording-ended.json` o `recording-failed.json`.

Estos contienen metadatos relacionados con la sesión grabada y sus formatos de salida. Los detalles de JSON se dan a continuación.

La carpeta `media` contiene el contenido multimedia compatible. La subcarpeta `hls` contiene todos los archivos multimedia y de manifiesto generados durante la sesión compuesta y se puede reproducir con el reproductor de IVS. El manifiesto HLS se encuentra en la carpeta `multivariant.m3u8`.

Política de buckets para StorageConfiguration

Cuando se crea un StorageConfiguration objeto, IVS tendrá acceso para escribir contenido en el bucket de S3 especificado. Este acceso se concede mediante modificaciones en la política del bucket de S3. Si la política del bucket se modifica de forma que se elimine el acceso del IVS, las grabaciones nuevas y en curso fallarán.

El siguiente ejemplo muestra una política de bucket de S3 que permite a IVS escribir en el bucket de S3:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CompositeWrite-y1d212y",
            "Effect": "Allow",
            "Principal": {
                "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::my-s3-bucket/*",
            "Condition": {
```

```
                "StringEquals": {
                    "s3:x-amz-acl": "bucket-owner-full-control"
                },
                "Bool": {
                    "aws:SecureTransport": "true"
                }
            }
        }
    ]
}
```

Archivos de metadatos JSON

Estos metadatos están en formato JSON. Contiene la siguiente información:

Campo	Tipo	Obligatorio	Descripción
`stage_arn`	string	Sí	ARN del escenario que se utiliza como fuente de la composición.
`media`	objeto	Sí	Objeto que contiene los objetos enumerados de contenido multimedia disponibles para esta grabación. Valores válidos: `"hls"`.
`hls`	objeto	Sí	Campo enumerado que describe la salida del formato HLS de Apple.
`duration_ms`	integer	Condicional	Duración del contenido HLS grabado en milisegundos. Esto solo está disponible cuando `recording_status` es `"RECORDING_ENDED"` o `"RECORDING_ENDED_WITH_FAILURE"`. Si se produjo un error antes de realizar cualquier grabación, esto es 0.

Campo	Tipo	Obligatorio	Descripción
`path`	string	Sí	Ruta relativa del prefijo S3 donde se almacena el contenido HLS.
`playlist`	string	Sí	Nombre del archivo de lista de reproducción maestra HLS.
`renditions`	objeto	Sí	Matriz de copias (variante HLS) de objetos de metadatos. Siempre hay al menos una copia.
`path`	string	Sí	Ruta relativa del prefijo S3 donde se almacena el contenido HLS para esta copia.
`playlist`	string	Sí	Nombre del archivo de lista de reproducción multimedia para esta copia.
`resolution_height`	int	Condicional	Altura de resolución de píxeles del video codificado. Solo está disponible cuando la copia contiene una pista de video.
`resolution_width`	int	Condicional	Ancho de resolución de píxeles del video codificado. Solo está disponible cuando la copia contiene una pista de video.

Campo	Tipo	Obligatorio	Descripción
`recording_ended_at`	cadena	Condicional	Marca de tiempo UTC RFC 3339 cuando finalizó la grabación . Esto solo está disponible cuando `recording_status` es `"RECORDING_ENDED"` o `"RECORDING_ENDED_WITH_FAILURE"` . `recording_started_at` y `recording_ended_at` son marcas de tiempo cuando se generan estos eventos y es posible que no coincidan exactamente con las marcas de tiempo del segmento de video HLS. Para determinar con precisión la duración de una grabación, utilice la `duration_ms` .
`recording_started_at`	cadena	Condicional	Marca de hora UTC RFC 3339 cuando se inició la grabación. No está disponible cuando `recording_status` está`RECORDING_START_FAILED` . Consulte la nota que figura arriba para `recording_ended_at` .

Campo	Tipo	Obligatorio	Descripción
`recording_status`	cadena	Sí	El estado de la grabación. Valores válidos: `"RECORDING_STARTED"`, `"RECORDING_ENDED"`, `"RECORDING_START_FAILED"`, `"RECORDING_ENDED_WITH_FAILURE"`.
`recording_status_message`	cadena	Condicional	Información descriptiva sobre el estado. Esto solo está disponible cuando `recording_status` es `"RECORDING_ENDED"` o `"RECORDING_ENDED_WITH_FAILURE"`.
`version`	cadena	Sí	La versión del esquema de metadatos.

Ejemplo: recording-started.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-11-01T06:01:36Z",
  "recording_status": "RECORDING_STARTED",
  "media": {
    "hls": {
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
```

```
    }
  }
}
```

Ejemplo: recording-ended.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-10-27T17:00:44Z",
  "recording_ended_at": "2023-10-27T17:08:24Z",
  "recording_status": "RECORDING_ENDED",
  "media": {
    "hls": {
      "duration_ms": 460315,
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

Ejemplo: recording-failed.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-10-27T17:00:44Z",
  "recording_ended_at": "2023-10-27T17:08:24Z",
  "recording_status": "RECORDING_ENDED_WITH_FAILURE",
  "media": {
    "hls": {
      "duration_ms": 460315,
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
```

```
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

Reproducción de contenido grabado desde buckets privados

El contenido grabado es privado de forma predeterminada; por lo tanto, no se puede acceder a ellos para reproducirlos mediante la dirección URL directa de S3. Si intenta abrir la lista de reproducción multivariada HLS (archivo m3u8) para su reproducción mediante el reproductor de IVS u otro reproductor, recibirá un error (por ejemplo, “You do not have permission to access the requested resource” [No tiene permiso para acceder al recurso solicitado]). En su lugar, puede reproducir estos archivos con Amazon CloudFront CDN (Content Delivery Network).

CloudFront las distribuciones se pueden configurar para ofrecer contenido desde depósitos privados. Por lo general, esto es preferible a tener depósitos de acceso abierto en los que las lecturas eviten los controles ofrecidos por ellos. CloudFront Puedes configurar tu distribución para que se publique desde un depósito privado creando un control de acceso de origen (OAC), que es un CloudFront usuario especial que tiene permisos de lectura en el depósito de origen privado. Puedes crear el OAC después de crear tu distribución, a través de la CloudFront consola o la API. Consulta Cómo crear un nuevo control de acceso a Origin en la Guía para CloudFront desarrolladores de Amazon.

Configuración de la reproducción CloudFront con CORS activado

En este ejemplo, se explica cómo un desarrollador puede configurar una CloudFront distribución con CORS activado, lo que permite la reproducción de sus grabaciones desde cualquier dominio. Esto resulta especialmente útil durante la fase de desarrollo, pero puede modificar el siguiente ejemplo para adaptarlo a sus necesidades de producción.

Paso 1: crear un bucket de S3

Crear un bucket de S3 que se utilizará para almacenar las grabaciones. Tenga en cuenta que el bucket debe estar en la misma región que utiliza para su flujo de trabajo de IVS.

Agregar una política de permisos CORS al bucket:

1. En la consola de AWS, vaya a la pestaña Permisos de bucket de S3.
2. Copie la política de CORS que se encuentra a continuación y péguela en Cross-origin resource sharing (CORS). Esto habilitará el acceso a CORS en el bucket de S3.

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

Paso 2: Crear una distribución CloudFront

Consulte Creación de una CloudFront distribución en la Guía para CloudFront desarrolladores.

Mediante la consola de AWS, ingrese la siguiente información:

Para este campo…	Elija esto...
Dominio de origen	El bucket de S3 en el paso anterior.

Para este campo…	Elija esto...
Acceso de origen	Configuración del control de acceso de origen (recomendado), mediante parámetros por defecto
Comportamiento predeterminado de la caché: política de protocolo de visualización	Redireccionamiento de HTTP a HTTPS
Comportamiento predeterminado de la caché: métodos HTTP permitidos	GET, HEAD y OPTIONS
Comportamiento predeterminado de la caché: solicitudes de origen y clave de caché	CachingDisabled política
Comportamiento de caché predeterminado: política de solicitudes de Origin	CORS-S3Origin
Comportamiento predeterminado de la caché: política de encabezados de respuesta	SimpleCORS
firewall de aplicaciones web	Habilitar protecciones de seguridad

A continuación, guarde la CloudFront distribución.

Paso 3: configurar la política de bucket de S3

1. Elimine StorageConfiguration las que haya configurado para el bucket de S3. Esto eliminará todas las políticas de bucket que se hayan agregado automáticamente al crear la política para ese bucket.
2. Vaya a su CloudFront distribución, asegúrese de que todos los campos de distribución estén en los estados definidos en el paso anterior y copie la política de buckets (utilice el botón Copiar política).
3. Vaya a su bucket de S3. En la pestaña Permisos, seleccione Editar política de bucket y pegue la política de bucket que copió en el paso anterior. Después de este paso, la política de segmentos debería incluir exclusivamente esa CloudFront política.
4. Cree un StorageConfiguration, especificando el bucket de S3.

Una vez creado, verá dos elementos en la política de cubos de S3: uno que permite CloudFront leer el contenido y otro que permite a IVS escribir contenido. StorageConfiguration Un ejemplo de política de bucket final, con CloudFront acceso a IVS, se muestra en Ejemplo: Política de bucket de S3 con CloudFront acceso a IVS.

Paso 4: reproducir grabaciones

Tras configurar correctamente la CloudFront distribución y actualizar la política de compartimentos, debería poder reproducir las grabaciones con el reproductor IVS:

1. Inicie correctamente una composición y asegúrese de tener una grabación almacenada en el bucket de S3.
2. Tras seguir los pasos 1 a 3 de este ejemplo, los archivos de vídeo deberían estar disponibles para su consumo a través de la CloudFront URL. Tu CloudFront URL es el nombre del dominio de distribución que aparece en la pestaña Detalles de la CloudFront consola de Amazon. Debe tener un aspecto similar al siguiente:

   ```
   a1b23cdef4ghij.cloudfront.net
   ```
3. Para reproducir el vídeo grabado a través de la CloudFront distribución, busca la clave de objeto del `multivariant.m3u8` archivo en el compartimento s3. Debe tener un aspecto similar al siguiente:

   ```
   FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/
   multivariant.m3u8
   ```
4. Añade la clave de objeto al final de la CloudFront URL. Su URL final tendrá un aspecto similar al siguiente:

   ```
   https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/
   fjFKbylPb3k4/composite/media/hls/multivariant.m3u8
   ```
5. Ahora puede añadir la URL final al atributo fuente de un reproductor IVS para ver la grabación completa. Para ver el vídeo grabado, puede utilizar la demostración en Cómo empezar en el SDK del reproductor de IVS: Guía web.

Ejemplo: política de buckets de S3 con acceso CloudFront a IVS

El siguiente fragmento ilustra una política de bucket de S3 que CloudFront permite leer el contenido en el bucket privado e IVS escribir contenido en el bucket. Nota: No copie ni pegue el siguiente

fragmento en su propio bucket. Su política debe contener los ID que sean relevantes para su CloudFront distribución y. StorageConfiguration

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CompositeWrite-7eiKaIGkC9DO",
      "Effect": "Allow",
      "Principal": {
        "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        },
        "Bool": {
          "aws:SecureTransport": "true"
        }
      }
    },
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::844311324168:distribution/
E1NG4YMW5MN25A"
        }
      }
    }
  ]
}
```

Solución de problemas

- La composición no está escrita en el depósito de S3: asegúrese de que el depósito de S3 y StorageConfiguration los objetos estén creados y estén en la misma región. Asegúrese también de que IVS tenga acceso al bucket consultando su política de bucket; consulte la Política de bucket para StorageConfiguration.
- No encuentro una composición cuando toco ListCompositions: las composiciones son recursos efímeros. Una vez que pasan a un estado final, se eliminan automáticamente después de unos minutos.
- Mi composición se detiene automáticamente: una composición se detiene automáticamente si no hay ningún publicador en el escenario durante más de 60 segundos.

Problema conocido

La lista de reproducción multimedia escrita mediante grabación compuesta tiene la etiqueta `#EXT-X-PLAYLIST-TYPE:EVENT` mientras la composición está en curso. Una vez finalizada la composición, la etiqueta se actualiza a `#EXT-X-PLAYLIST-TYPE:VOD`. Para una experiencia de reproducción fluida, le recomendamos que utilice esta lista de reproducción solo después de que la composición se haya finalizado correctamente.

Soporte para OBS y WHIP (transmisión en tiempo real)

Este documento explica cómo utilizar codificadores compatibles con WHIP, como OBS, para publicar en IVS en tiempo real. WHIP (Protocolo de ingestión WebRTC-HTTP) es un borrador del IETF desarrollado para estandarizar la ingestión de WebRTC.

El WHIP permite la compatibilidad con software como OBS y ofrece una alternativa (al SDK de transmisión del IVS) para la autoedición. Es posible que los streamers más sofisticados estén familiarizados con OBS y lo prefieran por sus características de producción avanzadas, como las transiciones de escenas, la mezcla de audio y la superposición de gráficos. Esto ofrece a los desarrolladores una opción versátil: usar el SDK de transmisión web de IVS para publicar directamente en el navegador o permitir que los streamers usen OBS en sus computadoras de escritorio para obtener herramientas más potentes.

Además, WHIP resulta útil en situaciones en las que el uso del SDK de transmisión de IVS no es factible o no es preferible. Por ejemplo, en configuraciones que utilizan codificadores de hardware, es posible que el SDK de transmisión IVS no sea una opción. Sin embargo, si el codificador es compatible con WHIP, puede seguir publicando directamente desde el codificador en IVS.

Guía OBS

OBS es compatible con WHIP a partir de la versión 30. Para empezar, descargue OBS v30 o posterior: https://obsproject.com/.

Para publicar en un escenario de IVS mediante OBS mediante WHIP, sigue estos pasos:

1. Genere un token de participante con capacidad de publicación. En términos de WHIP, un token de participante es un token portador. De forma predeterminada, las fichas de participante caducan en 12 horas, pero puedes ampliar su duración hasta 14 días.
2. Haga clic en Settings (Configuración). En la sección Transmisión del panel de configuración, selecciona WHIP en el menú desplegable del Servicio.
3. Para el servidor, ingresa https://global.whip.live-video.net/.
4. Para el token de portador, introduce el token de participante que generaste en el paso 2.
5. Configura los ajustes de vídeo como lo harías normalmente, con algunas restricciones:
   a. La transmisión en tiempo real de IVS admite entradas de hasta 720p a 8,5 Mbps. Si superas alguno de estos límites, tu transmisión se desconectará.

b. Te recomendamos configurar el intervalo de fotogramas clave en el panel de salida en 1 o 2 segundos. Un intervalo de fotogramas clave bajo permite a los espectadores iniciar la reproducción del vídeo con mayor rapidez. También recomendamos configurar el ajuste preestablecido de uso de la CPU en ultrarrápido y ajustarlo a latencia cero para permitir la latencia más baja.

c. Como OBS no admite la transmisión simultánea, recomendamos mantener la velocidad de bits por debajo de 2,5 Mbps. Esto permite a los espectadores con conexiones de menor ancho de banda ver.

6. Pulsa Iniciar transmisión.

Service Quotas (transmisión en tiempo real)

A continuación, se indican las cuotas de servicio y los límites para los puntos de conexión en tiempo real, los recursos y otras operaciones de Amazon Interactive Video Service (IVS). Las cuotas de servicio (que también se denominan límites) establecen el número máximo de recursos u operaciones de servicio que puede haber en una cuenta de AWS. Es decir, estos límites corren por cuenta de AWS, a menos que se indique lo contrario en la tabla. Consulte también Service Quotas de AWS.

Para conectarse mediante programación a un servicio de AWS, utilice un punto de conexión. Consulte también Puntos de conexión del servicio de AWS.

Todas las cuotas se aplican por región.

Aumentos en la cuota de servicio

Para cuotas ajustables, puede solicitar un aumento de la tasa a través de la consola de AWS. También, utilice la consola para consultar información sobre cuotas de servicio.

Las cuotas de tarifa de llamadas de API no son ajustables.

Cuotas de tarifa de llamadas a la API

Tipo de punto de conexión	Punto de conexión	Predeterminado
Composición	`GetComposition`	5 TPS
Composición	`ListCompositions`	5 TPS
Composición	`StartComposition`	5 TPS
Composición	`StopComposition`	5 TPS
MediaEncoder	`CreateEncoderConfiguration`	5 TPS
MediaEncoder	`DeleteEncoderConfiguration`	5 TPS
MediaEncoder	`GetEncoderConfiguration`	5 TPS

Tipo de punto de conexión	Punto de conexión	Predeterminado
MediaEncoder	`ListEncoderConfigurations`	5 TPS
Escenario	`CreateParticipantToken`	50 TPS
Escenario	`CreateStage`	5 TPS
Escenario	`DeleteStage`	5 TPS
Escenario	`DisconnectParticipant`	5 TPS
Escenario	`GetParticipant`	5 TPS
Escenario	`GetStage`	5 TPS
Escenario	`GetStageSession`	5 TPS
Escenario	`ListStages`	5 TPS
Escenario	`UpdateStage`	5 TPS
Escenario	`ListParticipants`	5 TPS
Escenario	`ListParticipantEvents`	5 TPS
Escenario	`ListStageSessions`	5 TPS
StorageConfiguration	`CreateStorageConfiguration`	5 TPS
StorageConfiguration	`DeleteStorageConfiguration`	5 TPS
StorageConfiguration	`GetStorageConfiguration`	5 TPS
StorageConfiguration	`ListStorageConfigurations`	5 TPS
Etiquetas	`ListTagsForResource`	10 TPS
Etiquetas	`TagResource`	10 TPS
Etiquetas	`UntagResource`	10 TPS

Otras cuotas

Recurso o característica	Predeterminado	Ajustable	Descripción
EncoderConfigurations	20	Sí	Número máximo de recursos de configuración de codificadores por cuenta.
Destinos de composición	2	No	Número máximo de objetos de destino en un recurso de composición.
Composición: duración máxima	24	No	Cantidad máxima de tiempo que puede existir una composición, en horas.
Composiciones	5	Sí	Cantidad máxima de recursos de composición simultáneos por cuenta.
Duración de la publicación o suscripción de los participantes	24	No	Periodo máximo de tiempo durante el cual un participante puede publicar o permanecer suscrito a un escenario, en horas.
El participante publica la resolución	720p	No	Resolución máxima del video publicado por los participantes.
Tasa de bits de descarga de participantes	8,5 Mbps	No	Velocidad máxima de bits de descarga agregada en todas las suscripciones de un participante.

Recurso o característica	Predeterminado	Ajustable	Descripción
Participantes de la fase (publicadores)	12	No	Número máximo de participantes que pueden publicar en una fase a la vez.
Participantes de la fase (suscriptores)	10 000	Sí	Número máximo de participantes que pueden suscribirse a una fase a la vez.
Escenarios	100	Sí	Número máximo de escenarios por región de AWS.

Optimizaciones de transmisión en tiempo real

Para garantizar que sus usuarios tengan la mejor experiencia al transmitir y ver videos mediante la transmisión en tiempo real de IVS, hay varias formas de mejorar u optimizar partes de la experiencia con las características que ofrecemos actualmente.

Introducción

Al optimizar la calidad de la experiencia de un usuario, es importante tener en cuenta la experiencia deseada, que puede cambiar según el contenido que esté viendo y las condiciones de la red.

A lo largo de esta guía, nos centramos en los usuarios que son publicadores de transmisiones o suscriptores de transmisiones, y tenemos en cuenta las acciones y experiencias deseadas de esos usuarios.

Transmisión adaptativa: codificación en capas con transmisión simultánea

Esta característica solo se admite en las siguientes versiones de cliente:

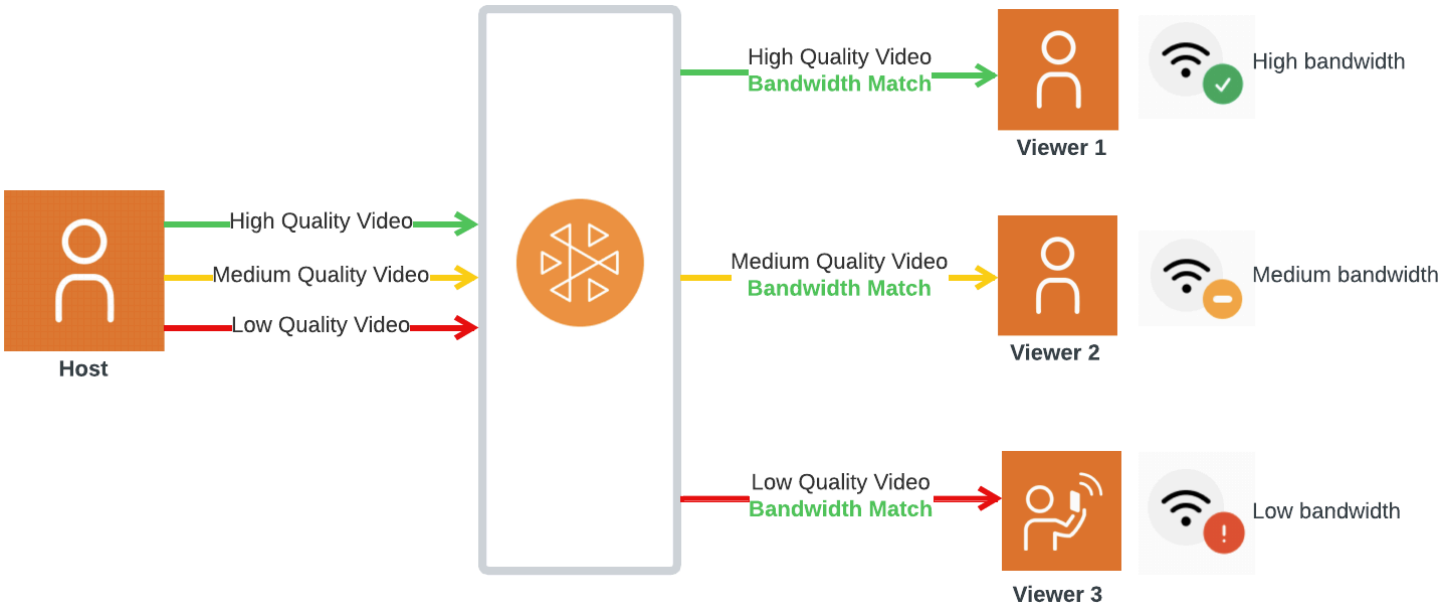
- iOS y Android 1.12.0 +
- Web 1.5.1 +

Debe enviar un correo electrónico a amazon-ivs-simulcast@amazon .com para habilitar esta función en su cuenta. La activación de la transmisión simultánea a través de la configuración del SDK no tendrá ningún efecto, a menos que haya optado por ello.

Una vez que ha optado por la característica, cuando se usan los SDK de transmisión en tiempo real de IVS, los publicadores codifican varias capas de video y los suscriptores se adaptan o cambian automáticamente a la calidad que mejor se adapte a su red. A esto lo llamamos codificación en capas con transmisión simultánea.

La codificación en capas con transmisión simultánea es compatible con Android y iOS, y con los navegadores de escritorio Chrome (para Windows y macOS). No admitimos la codificación en capas en otros navegadores.

En el siguiente diagrama, el host envía tres calidades de video (alta, media y baja). IVS reenvía video de la más alta calidad a cada espectador en función del ancho de banda disponible. Esto proporciona una experiencia óptima para cada espectador. Si la conexión de red del espectador 1 cambia de buena a mala, IVS comienza automáticamente a enviarle video de menor calidad, de modo que puede seguir viendo la transmisión sin interrupciones (con la mejor calidad posible).



Capas, calidades y velocidades de fotogramas predeterminadas

Las calidades y capas predeterminadas que se proporcionan a los usuarios de dispositivos móviles y de la web son las siguientes:

Móvil (Android, iOS)	Web (Chrome)
Capa alta (o personalizada): • Velocidad de bits máxima: 900 000 bps • Velocidad de fotogramas: 15 fps • Resolución: 360x640	Capa alta (o personalizada): • Velocidad de bits máxima: 1 700 000 bps • Velocidad de fotogramas: 30 fps • Resolución: 1280x720
Capa intermedia: ninguna (no es necesaria, porque la diferencia entre las velocidades de bits de capa alta	Capa intermedia: • Velocidad de bits máxima: 700 000 bps • Velocidad de fotogramas: 20 fps

Móvil (Android, iOS)	Web (Chrome)
y baja en los dispositivos móviles es estrecha)	• Resolución: 640x360
Capa baja: • Velocidad de bits máxima: 150 000 bps • Velocidad de fotogramas: 15 fps • Resolución: 180x320	Capa baja: • Velocidad de bits máxima: 200 000 bps • Velocidad de fotogramas: 15 fps • Resolución: 320x180

Configuración de la codificación en capas con transmisión simultánea

Para utilizar la codificación por capas con transmisión simultánea, debe haber optado por esta función y haberla activado en el cliente. Si la habilita, verá un aumento en la velocidad de bits general transmitida, con la ventaja de que el vídeo se congelará menos.

Android

```
// Opt-out of Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
// Opt-out of Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

let cameraStream = IVSLocalStageStream(device: camera, configuration: config)

// Other Stage implementation code
```

Web

```
// Opt-out of Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})

// Other Stage implementation code
```

Configuraciones de transmisión

En esta sección, se exploran otras configuraciones que puede aplicar a las transmisiones de video y audio.

Cambiar la velocidad de bits de la transmisión

Para cambiar la velocidad de bits de la transmisión, use los siguientes ejemplos de configuración.

Android

```
StageVideoConfiguration config = new StageVideoConfiguration();

// Update Max Bitrate to 1.5mbps
config.setMaxBitrate(1500000);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamVideoConfiguration();

// Update Max Bitrate to 1.5mbps
try! config.setMaxBitrate(1500000);

let cameraStream = IVSLocalStageStream(device: camera, configuration: config);

// Other Stage implementation code
```

Web

```
// Note: On web it is also recommended to configure the framerate of your device from
 userMedia
const camera = await navigator.mediaDevices.getUserMedia({
   video: {
      bitrate: {
         ideal: 1500,
         max: 1500,
      },
   },
});

let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Bitrate to 1.5mbps or 1500kbps
   maxBitrate: 1500
})

// Other Stage implementation code
```

Cambiar la velocidad de fotogramas de la transmisión de video

Para cambiar la velocidad de fotogramas de la transmisión de video, use los siguientes ejemplos de configuración.

Android

```
StageVideoConfiguration config = new StageVideoConfiguration();

// Update target framerate to 10fps
config.targetFramerate(10);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamVideoConfiguration();

// Update target framerate to 10fps
try! config.targetFramerate(10);

let cameraStream = IVSLocalStageStream(device: camera, configuration: config);
```

```
// Other Stage implementation code
```

Web

```
// Note: On web it is also recommended to configure the framerate of your device from
 userMedia
const camera = await navigator.mediaDevices.getUserMedia({
   video: {
      frameRate: {
         ideal: 10,
         max: 10,
      },
   },
});

let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Framerate to 10fps
   maxFramerate: 10
})
// Other Stage implementation code
```

Optimización de la tasa de bits de audio y el soporte estéreo

Para cambiar la velocidad de bits y la configuración estéreo de la transmisión de audio, use los siguientes ejemplos de configuración.

Web

```
// Note: Disable autoGainControl, echoCancellation, and noiseSuppression when enabling
 stereo.
const camera = await navigator.mediaDevices.getUserMedia({
   audio: {
      autoGainControl: false,
      echoCancellation: false,
      noiseSuppression: false
   },
});

let audioStream = new LocalStageStream(camera.getAudioTracks()[0], {
   // Optional: Update Max Audio Bitrate to 96Kbps. Default is 64Kbps
   maxAudioBitrateKbps: 96,
```

```
   // Signal stereo support. Note requires dual channel input source.
   stereo: true
})

// Other Stage implementation code
```

Android

```
StageAudioConfiguration config = new StageAudioConfiguration();

// Update Max Bitrate to 96Kbps. Default is 64Kbps.
config.setMaxBitrate(96000);

AudioLocalStageStream microphoneStream = new AudioLocalStageStream(microphone, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamConfiguration();

// Update Max Bitrate to 96Kbps. Default is 64Kbps.
try! config.audio.setMaxBitrate(96000);

let microphoneStream = IVSLocalStageStream(device: microphone, config: config);

// Other Stage implementation code
```

Optimizaciones sugeridas

Escenario	Recomendaciones
Transmite con texto o contenido que se mueve lentamente, como presentaciones o diapositivas	Utilice la codificación por capas con la transmisión simultánea o configure las transmisiones con una velocidad de fotogramas más baja.
Transmisiones con acción o mucho movimiento	Utilice la codificación por capas con la transmisión simultánea.

Escenario	Recomendaciones
Transmisiones con conversación o poco movimiento	Utilice la codificación por capas con la transmisión simultánea o elija solo audio (consulte la sección “Suscripción a los participantes” en las guías del SDK de transmisiones en tiempo real: Web, Android e iOS).
Usuarios que transmiten con datos limitados	Utilice la codificación por capas con transmisión simultánea o, si desea reducir el uso de datos para todos, configure una velocidad de fotogramas más baja y baje la velocidad de bits manualmente.

Recursos y asistencia (streaming en tiempo real)

Recursos

https://ivs.rocks/ es un sitio dedicado a explorar el contenido publicado (demostraciones, ejemplos de código, publicaciones de blog), calcular el costo y experimentar Amazon IVS a través de demostraciones en vivo.

Demostraciones

La demostración de streaming en tiempo real de IVS para iOS y Android muestra a los desarrolladores cómo utilizar Amazon IVS para crear una atractiva aplicación de contenido en tiempo real generada por usuarios de redes sociales. Esta aplicación incluye una fuente desplazable de transmisiones en tiempo real generadas por los usuarios. Los usuarios pueden crear transmisiones de video y salas solo de audio. Los invitados que hagan streaming de video pueden unirse en modo

invitado o en modo versus (VS). Las instrucciones sobre cómo implementar el backend requerido y crear la aplicación están disponibles en los siguientes repositorios de GitHub:

- iOS: https://github.com/aws-samples/amazon-ivs-real-time-for-ios-demo/
- Android: https://github.com/aws-samples/amazon-ivs-real-time-for-android-demo/
- Backend: https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

Soporte

El Centro de AWS ofrece una serie de planes que proporcionan acceso a herramientas y conocimientos para dar soporte a sus soluciones de AWS. Todos los planes de soporte proporcionan acceso ininterrumpido a atención al cliente. Para obtener asistencia técnica y recursos adicionales para planificar, implementar y mejorar su entorno de AWS, seleccione un plan de asistencia técnica acorde con su caso de uso de AWS.

AWS Premium Support es un canal de asistencia individualizado y de respuesta rápida que le ayudará a crear y ejecutar aplicaciones en AWS.

AWS re:Post es un sitio de preguntas y respuestas para la comunidad de desarrolladores donde se tratan aspectos técnicos relacionados con Amazon IVS.

En Contacte con nosotros tiene enlaces para hacernos llegar sus preguntas sobre facturación o su cuenta. Para preguntas técnicas, utilice los foros de debate o los enlaces de soporte previamente proporcionados.

Glosario

Consulte también el glosario de AWS. En la siguiente tabla, LL son las siglas de transmisión de baja latencia de IVS; RT, transmisión en tiempo real de IVS.

Plazo	Descripción	LL	RT	Chat
AAC	Advanced Audio Coding. AAC es un estándar de codificación de audio para la compresión de audio digital con pérdida. Diseñado para ser el sucesor del formato MP3, AAC generalmente logra una calidad de sonido superior a este con la misma velocidad de bits. El formato AAC fue estandarizado por la ISO y la IEC como parte de las especificaciones MPEG-2 y MPEG-4.	✓	✓	
Transmisión con velocidad de bits adaptable	La transmisión con velocidad de bits adaptable (ABR) permite al reproductor de IVS cambiar a una velocidad de bits más baja cuando la calidad de la conexión se vea afectada y volver a una velocidad más alta cuando la calidad mejore.	✓		
Transmisión adaptativa	Consulte Codificación por capas con transmisión simultánea.		✓	
Usuario administrativo	Un usuario de AWS con acceso administrativo a los recursos y servicios disponibles en una cuenta de AWS. Consulte Terminología en la Guía del usuario de configuración de AWS.	✓	✓	✓
ARN	Nombre de recurso de Amazon, un identificador exclusivo de cualquier recurso de AWS. Los formatos específicos de ARN dependen del tipo de recurso. Para conocer los formatos de ARN utilizados por los recursos de IVS, consulte la referencia de autorización de servicio.	✓	✓	✓

Plazo	Descripción	LL	RT	Chat
Relación de aspecto	Describe la relación entre el ancho y la altura del marco. Por ejemplo, 16:9 es la relación de aspecto que corresponde a la resolución Full HD o 1080p.	✓	✓	
Modo de audio	Una configuración de audio preestablecida o personalizada que está optimizada para diferente s tipos de usuarios de dispositivos móviles y los equipos que usan. Consulte SDK de transmisión de IVS: modos de audio móvil (streaming en tiempo real).		✓	
AVC, H.264, MPEG-4 parte 10	La codificación de video avanzada, también denominada H.264 o MPEG-4 parte 10, es un estándar para la compresión de video digital con pérdida.	✓	✓	
Reemplazo de fondo	Un tipo de filtro de cámara que permite a los creadores de transmisiones en directo cambiar sus fondos. Consulte Reemplazo de fondo en SDK de transmisión de IVS: filtros de cámara de terceros (streaming en tiempo real).		✓	
Velocidad de bits	Métrica de transmisión que indica el número de bits que se trasmiten o reciben por segundo.	✓	✓	
Difusión, emisora	Otros términos para transmisión o streamer.	✓		

Plazo	Descripción	LL	RT	Chat
Almacenamiento en búfer	Una condición que se produce cuando el dispositivo de reproducción no puede descargar el contenido antes de que este se reproduzca. El almacenamiento en búfer puede manifestarse de varias formas: el contenido puede detenerse y comenzar de forma aleatoria (lo que también se conoce como inestabilidad), el contenido puede detenerse durante periodos prolongados (también conocido como congelación) o el reproductor de IVS se puede pausar.	✓	✓	
Listas de reproducción por rango de bytes	Una lista de reproducción más detallada que la lista de reproducción HLS estándar. La lista de reproducción HLS estándar se compone de archivos multimedia de 10 segundos. Con una lista de reproducción por rango de bytes, la duración del segmento es la misma que el intervalo de fotogramas clave configurado para la transmisión. La lista de reproducción por rango de bytes solo está disponible para las emisiones que se grabaron automáticamente en un bucket de S3. Se crea además de la lista de reproducción HLS. Consulte Listas de reproducción por rango de bytes en Grabación automática en Amazon S3 (streaming de baja latencia).	✓		

Plazo	Descripción	LL	RT	Chat
CBR	Velocidad de bits constante (CBR): un método de control de velocidad para codificadores que mantiene una velocidad de bits constante durante toda la reproducción de un video, sin importar lo que ocurra durante la transmisión. Las pausas en la acción se pueden rellenar para alcanzar la velocidad de bits deseada. De igual manera, los picos se pueden cuantificar al ajustar la calidad de la codificación para que coincida con la velocidad de bits objetivo. Recomendamos encarecidamente utilizar CBR en lugar de VBR.	✓	✓	
CDN	Red de entrega de contenidos o red de distribución de contenidos: una solución distribuida de manera geográfica que optimiza la distribución de contenidos, como la transmisión de video, y la acerca al lugar donde se encuentran los usuarios.	✓		
Canal	Un recurso de IVS que almacena la configuración para la transmisión, lo que incluye un servidor de ingesta, una clave de transmisión, una URL de reproducción y opciones de grabación. Los streamers utilizan la clave de transmisión asociada a un canal para iniciar una difusión. Todas las métricas y eventos están asociados a un recurso de canal.	✓		
Tipo de canal	Determina la resolución y la velocidad de fotogramas permitidas para el canal. Consulte Tipos de canales en la Referencia de la API de transmisión de baja latencia de IVS.	✓		
Registro del chat	Una opción avanzada que se puede habilitar mediante la vinculación de una configuración de registro con una sala de chat.			✓

Plazo	Descripción	LL	RT	Chat
Sala de chat	Recurso de IVS que almacena la configuración de una sesión de chat e incluye funciones opcionales como el controlador de revisión de mensajes y el registro de conversaciones. Consulte Step 2: Create a Chat Room en Getting Started with IVS Chat.			✓
Composición del cliente	Usa un dispositivo host para mezclar las transmisiones de audio y video de los participantes del escenario y, a continuación, las envía como una transmisión compuesta a un canal de IVS. Esto permite un mayor control sobre el aspecto de la composición, pero a costa de una mayor utilización de los recursos del cliente y un mayor riesgo de que un problema con el escenario o el anfitrión afecte a los espectadores. Consulte también Composición del servidor.	✓	✓	
CloudFront	Un servicio de CDN que proporciona Amazon.	✓		
CloudTrail	Un servicio de AWS para recopilar, supervisar, analizar y retener los eventos y la actividad de las cuentas de AWS, así como de orígenes externos. Consulte Registrar llamadas a la API de IVS con AWS CloudTrail.	✓	✓	✓
CloudWatch	Un servicio de AWS para supervisar las aplicaciones, responder a los cambios de rendimiento, optimizar el uso de los recursos y proporcionar información sobre el estado de las operaciones. Puede utilizarla CloudWatch para monitorizar las métricas de IVS; consulte Supervisión de la transmisión de IVS en tiempo real y Supervisión de la transmisión de baja latencia de IVS.	✓	✓	✓

Plazo	Descripción	LL	RT	Chat
Composición	Proceso que consiste en combinar transmisiones de audio y video de varios orígenes en una sola transmisión.	✓	✓	
Canalización de composición	Secuencia de pasos de procesamiento necesaria para combinar varias transmisiones y codificar la resultante.	✓	✓	
Compresión	Codificación de la información con menos bits que la representación original. Cualquier compresió n en particular puede ser con o sin pérdida. La compresión sin pérdida reduce los bits al identificar y eliminar la redundancia estadística. No se pierde información en la compresión sin pérdida. La compresión con pérdida reduce los bits al eliminar información innecesaria o menos importante.	✓	✓	
Plano de control	Almacena información sobre los recursos de IVS, como canales, escenarios o salas de chat. Además, proporciona interfaces para crear y administrar estos recursos. Es regional (se basa en las regiones de AWS).	✓	✓	✓
CORS	Uso compartido de recursos entre orígenes: una característica que permite a las aplicaciones web de los clientes cargadas en un dominio interactu ar con los recursos (como buckets de S3) de un dominio diferente. El acceso se puede configurar en función de los encabezados, los métodos HTTP y los dominios de origen. Consulte Uso compartid o de recursos entre orígenes (CORS): Amazon Simple Storage Service en la Guía del usuario de Amazon Simple Storage Service.	✓		

Plazo	Descripción	LL	RT	Chat
Origen de imagen personalizado	Una interfaz proporcionada por el SDK de transmisión de IVS que permite que una aplicación proporcione su propia entrada de imagen en vez de limitarse a las cámaras predeterminadas.	✓	✓	
Plano de datos	La infraestructura que transmite los datos desde la ingesta hasta la salida. Funciona según la configuración administrada en el plano de control y no está restringida a una región de AWS.	✓	✓	✓
Codificador, codificación	Proceso de convertir contenido de video y audio a un formato digital que sea adecuado para la transmisión. La codificación puede estar basada en hardware o software.	✓	✓	
Evento	Una notificación automática publicada por IVS al servicio de monitorización. AmazonEventBridge Un evento representa un cambio de estado de un recurso de transmisión, como un escenario o una canalización de composición. Consulte Uso de Amazon EventBridge con transmisión de baja latencia de IVS y Uso de Amazon EventBridge con transmisión en tiempo real de IVS.	✓	✓	✓
FFmpeg	Un proyecto de software libre y de código abierto que consiste en un conjunto de bibliotecas y programas para gestionar archivos y transmisiones de audio y video. FFmpeg proporciona una solución compatible con diversas plataformas para grabar, convertir y transmitir audio y video.	✓		

Plazo	Descripción	LL	RT	Chat
Transmisión fragmentada	Se crea cuando una transmisión se desconecta y, a continuación, se vuelve a conectar dentro del intervalo especificado en la configuración de grabación del canal. Los diversas transmisiones resultantes se consideran una sola, por lo que se combinan en una sola transmisión grabada. Consulte Fusionar transmisiones fragmentadas en Grabación automática en Amazon S3 (streaming de baja latencia).	✓		
Velocidad de fotogramas	Métrica de transmisión que indica el número de fotogramas que se trasmiten o reciben por segundo.	✓	✓	
HLS	HTTP Live Streaming (HLS): un protocolo de comunicaciones de transmisión con velocidad de bits adaptable que se basa en HTTP y se utiliza para entregar transmisiones de IVS a los espectadores.	✓		
Lista de reproducción HLS	Lista de segmentos multimedia que componen una transmisión. Las listas de reproducción HLS estándar se componen de archivos multimedia de 10 segundos. HLS también admite listas de reproducción con un rango de bytes más granular.	✓		
Host	Participante de un evento en tiempo real que envía video o audio al escenario.		✓	
IAM	Identity and Access Management: un servicio de AWS que permite a los usuarios administrar de forma segura las identidades y el acceso a los servicios y recursos de AWS, incluido IVS.	✓	✓	✓

Plazo	Descripción	LL	RT	Chat
Incorporación	Proceso de IVS que consiste en recibir transmisiones de video de un host o difusor para su procesamiento o entrega a los espectadores u otros participantes.	✓	✓	
Servidor de incorporación	Recibe transmisiones de video y las envía a un sistema de transcodificación, en el que las transmisiones se multiplexan o transcodifican en HLS para su entrega a los espectadores. Los servidores de ingesta son componentes específicos de IVS y reciben las transmisiones de los canales, así como un protocolo de ingesta (RTMP, RTMPS). Consulte la información sobre cómo crear un canal en Introducción al streaming de baja latencia de IVS.		✓	
Video entrelazado	Transmite y muestra solo las líneas pares o impares de los fotogramas posteriores para duplicar la velocidad de fotogramas percibida sin consumir más ancho de banda. No recomendamos utilizar video entrelazado debido a problemas en su calidad.	✓	✓	
JSON	JavaScript Notación de objetos, un formato de archivo de estándar abierto que utiliza texto legible por humanos para transmitir objetos de datos compuestos por pares de atributo-valor y tipos de datos de matriz u otros valores serializables.	✓	✓	✓

Plazo	Descripción	LL	RT	Chat
Fotograma clave, fotograma delta, intervalo de fotogramas clave	El fotograma clave (también denominado intracodificado o fotograma i) es un fotograma completo de la imagen de un video. Los fotogramas posteriores, los fotogramas delta (también denominados fotogramas previstos o fotogramas p), solo contienen la información que ha cambiado. Los fotogramas clave aparecerán varias veces dentro de una transmisión, según el intervalo de fotogramas clave definido en el codificador.	✓	✓	
Lambda	Un servicio de AWS para ejecutar código (denominadas funciones de Lambda) sin aprovisionar ninguna infraestructura de servidor. Las funciones de Lambda se pueden ejecutar en respuesta a eventos y solicitudes de invocación o según una programación. Por ejemplo, Chat de IVS utiliza funciones de Lambda para permitir la revisión de mensajes en una sala de chat.	✓	✓	✓
Latencia, latencia glass-to-glass	Un retraso en la transferencia de datos. IVS define los rangos de latencia de la siguiente manera: • Baja latencia: menos de 3 segundos • Latencia en tiempo real: menos de 300 ms La lass-to-glass latencia G se refiere al retraso que transcurre entre el momento en que una cámara captura una transmisión en directo y el momento en que la transmisión aparece en la pantalla del espectador.	✓	✓	

Plazo	Descripción	LL	RT	Chat
Codificación por capas con la transmisión simultánea	Permite la codificación y publicación simultáneas de varias transmisiones de video con distintos niveles de calidad. Consulte Transmisión adaptativa: codificación en capas con transmisión simultánea y Optimizaciones de streaming en tiempo real.		✓	
Controlador de revisión de mensajes	Permite a los clientes de Chat de IVS revisar y filtrar automáticamente los mensajes de los usuarios antes de enviarlos a la sala de chat. Se habilita al asociar una función de Lambda con una sala de chat. Consulte Creating a Lambda Function en Chat Message Review Handler.			✓
Mezclador	Una característica de los SDK de transmisión móvil de IVS que recibe varios orígenes de audio y video y genera una única salida. Admite la administración de los elementos de video y audio que están en pantalla y que representan orígenes como cámaras, micrófonos, capturas de pantalla, así como audio y video generados por la aplicación. Luego, la salida se puede transmitir a IVS. Consulte Configuración de una sesión de transmisión de mezcla en SDK de transmisión de IVS: guía del mezclador (streaming de baja latencia).	✓		
Transmisión de varios hosts	Combina transmisiones de varios hosts en una sola transmisión. Esto se puede lograr mediante una composición del cliente o del servidor. La transmisión de varios hosts permite, por ejemplo, invitar a los espectadores a un escenario para una sesión de preguntas y respuestas, concursos entre hosts, videoconferencias y hosts conversando entre sí frente a un gran público.		✓	

Plazo	Descripción	LL	RT	Chat
Lista de reproducción multivariante	Un índice de todas las transmisiones variantes disponibles para una transmisión.	✓		
OAC	Origin Access Control, un mecanismo para restringir el acceso a un depósito de S3, de modo que el contenido, como una transmisión grabada, solo se pueda publicar a través de la CloudFrontCDN.	✓		
OBS	Open Broadcaster Software: software gratuito y de código abierto para grabación y transmisión en directo de video. OBS ofrece una alternativa (al SDK de transmisión de IVS) para la autoedición. Es posible que los streamers más sofisticados estén familiarizados con OBS y lo prefieran por sus características de producción avanzadas, como las transiciones de escenas, la mezcla de audio y la superposición de gráficos.	✓	✓	
Participante	Un usuario en tiempo real conectado al escenario como host o espectador.		✓	
Token de participante	Autentica a un participante del evento en tiempo real cuando se une a un escenario. Un token de participante también controla si un participante puede enviar video al escenario.		✓	

Plazo	Descripción	LL	RT	Chat
Token de reproducción, par de claves de reproducción	Un mecanismo de autorización que permite a los clientes restringir la reproducción de video en canales privados. Los tokens de reproducción se generan a partir de un par de claves de reproducción. Un par de claves de reproducción es el par de claves público-privadas utilizado para firmar y validar el token de autorización del espectador para la reproducción. Consulte Crear o importar una clave de reproducción en Configuración de canales privados y consulte los puntos de conexión del par de claves de reproducción en la referencia de la API de baja latencia de IVS.	✓		
URL de reproducción	Identifica la dirección que utiliza el espectador para iniciar la reproducción de un canal específico. Esta dirección se puede utilizar globalmente. IVS selecciona automáticamente la mejor ubicación de la red global de distribución de contenido de IVS para entregar el video a cada espectador. Consulte la información sobre cómo crear un canal en Introducción al streaming de baja latencia de IVS.	✓		
Canal privado	Permite a los clientes restringir el acceso a sus transmisiones mediante un mecanismo de autorización que se basa en los tokens de reproducción. Consulte Flujo de trabajo para canales privados en Configuración de canales privados.	✓		

Plazo	Descripción	LL	RT	Chat
Video progresivo	Transmite y muestra todas las líneas de cada fotograma en secuencia. Recomendamos utilizar video progresivo durante todas las etapas de la transmisión.	✓	✓	
Cuotas	El número máximo de recursos u operaciones de servicio de IVS que puede haber en una cuenta de AWS. Es decir, estos límites corren por cuenta de AWS, a menos que se indique lo contrario. Todas las cuotas se aplican por región. Consulte Puntos de conexión y cuotas de Amazon Interactive Video Service en la Guía de referencia general de AWS.	✓	✓	✓
Regiones	Otorga acceso a servicios de AWS que se ubican físicamente en un área geográfica determinada. Las regiones proporcionar tolerancia a errores, estabilidad y resistencia, y también pueden reducir la latencia. Con las regiones, puede crear recursos redundantes que sigan estando disponibles y no resulten afectados por una interrupción regional. La mayoría de las solicitudes de servicio de AWS están asociadas a una región geográfica en particular. Los recursos que crea en una región no existen en ninguna otra región salvo que utilice explícitamente una característica de replicación ofrecida por un servicio de AWS. Por ejemplo, Amazon S3 admiten la replicación entre regiones. Algunos servicios, como IAM, no tienen recursos entre regiones.	✓	✓	✓
Resolución	Describe la cantidad de píxeles en un único fotograma de video; por ejemplo, Full HD o 1080p definen un fotograma con 1920 x 1080 píxeles.	✓	✓	

Plazo	Descripción	LL	RT	Chat
Usuario raíz	El propietario de una cuenta de AWS. Este usuario raíz tiene acceso a todos los recursos y los servicios de AWS en la cuenta de AWS.	✓	✓	✓
RTMP, RTMPS	Protocolo de mensajes en tiempo real: un estándar del sector para la transmisión de audio, video y datos a través de una red. RTMPS es la versión segura de RTMP, que se ejecuta a través de una conexión de seguridad de la capa de transporte (TLS/SSL).	✓	✓	
Bucket de S3	Un conjunto de objetos almacenados en Amazon S3. Muchas políticas, como las de acceso y replicación, se definen en el nivel de bucket y se aplican a todos los objetos del bucket. Por ejemplo, una transmisión de IVS se almacena como varios objetos en un bucket de S3.	✓		
SDK	Kit de desarrollo de software: un conjunto de bibliotecas para los desarrolladores que crean aplicaciones con IVS.	✓	✓	✓
Segmentación de selfies	Permite sustituir el fondo en una transmisión en directo mediante una solución específica para el cliente que acepta una imagen de la cámara como entrada y devuelve una máscara que proporcio na una puntuación de confianza para cada píxel de la imagen, que indica si está en primer plano o en segundo. Consulte Reemplazo de fondo en SDK de transmisión de IVS: filtros de cámara de terceros (streaming en tiempo real).		✓	

Plazo	Descripción	LL	RT	Chat
Control de versiones semántico	Formato de versión con el formato VersiónPrincipal.VersiónSecundaria.Revisión. Las correcciones de errores que no afectan a la API aumentan la versión de revisión, las adiciones o cambios de la API compatibles con versiones anteriores aumentan la versión secundaria y los cambios de la API incompatibles con versiones anteriores aumentan la versión principal.	✓	✓	✓
Composición del servidor	Utiliza un servidor de IVS para mezclar el audio y el video de todos los participantes del escenario y, a continuación, envía este video mezclado a un canal de IVS (por ejemplo, para llegar a un público más amplio) o a un bucket de S3. La composición del servidor reduce la carga de clientes, mejora la resiliencia de la transmisión y permite un uso más eficiente del ancho de banda. Consulte también Composición del cliente.		✓	
Cuotas de servicio	Un servicio de AWS que lo ayuda a administrar las cuotas de muchos servicios de AWS desde una sola ubicación. Además de buscar los valores de cuota, también puede solicitar un aumento de cuota desde la consola de Service Quotas.	✓	✓	✓
Rol vinculado a servicio	Un tipo único de rol de IAM que está vinculado directamente a un servicio de AWS. IVS crea automáticamente los roles vinculados a servicios e incluyen todos los permisos que el servicio requiere para llamar a otros servicios de AWS en su nombre, por ejemplo, a un bucket de S3. Consulte Uso de roles vinculados a servicios para IVS en Seguridad de IVS.	✓		

Plazo	Descripción	LL	RT	Chat
Escenario	Un recurso de IVS que representa un espacio virtual en el que los participantes del evento en tiempo real pueden intercambiar videos también en tiempo real. Consulte Creación de un escenario en Introducción a streaming en tiempo real de IVS.		✓	
Sesión de escenario	La sesión de un escenario comienza cuando el primer participante se une a un escenario y finaliza unos minutos después de que el último participa nte deje de publicar en él. Un escenario de larga duración puede tener varias sesiones a lo largo de su vida útil.		✓	
De streaming	Datos que representan contenido de video o audio que se envían de forma continua de un origen a un destino.	✓	✓	
Clave de transmisi ón	Identificador asignado por IVS cuando se crea un canal y se utiliza para autorizar la transmisión. Trate la clave de transmisión como un secreto, ya que permite a cualquiera que la tenga transmitir en el canal. Consulte Introducción al streaming de baja latencia de IVS.	✓		

Plazo	Descripción	LL	RT	Chat
Transmisión en inanición	Un retraso o una interrupción en la entrega de la transmisión a IVS. Se produce cuando IVS no recibe la cantidad esperada de bits que el dispositivo de codificación anunció que enviaría en un periodo determinado. Si se produce una transmisión en inanición, se produce un evento de transmisión en inanición. Desde la perspectiva del espectador, la transmisión en inanición puede manifestarse como un video que se retrasa, se almacena en búfer o se congela. La transmisión en inanición puede ser breve (menos de 5 segundos) o larga (varios minutos), según la situación específica que la provocó. Consulte ¿Qué es la transmisión en inanición? en Preguntas frecuentes de solución de problemas.	✓	✓	
Streamer	Persona o dispositivo que envía una transmisión de video o audio a IVS.	✓	✓	
Suscriptor	Participante de un evento en tiempo real que recibe video o audio del host. Consulte ¿Qué es el streaming en tiempo real de IVS?.		✓	
Etiqueta	Una etiqueta de metadatos que asigna a un recurso de AWS. Las etiquetas pueden ayudarlo a identificar y organizar sus recursos de AWS. En la página de inicio de la documentación de IVS, consulte la sección “Etiquetado” en cualquier documentación de la API de IVS (para la transmisión en tiempo real, la transmisión de baja latencia o el chat).	✓	✓	✓

Plazo	Descripción	LL	RT	Chat
Filtros de cámara de terceros	Componentes de software que se pueden integrar con el SDK de transmisión de IVS para permitir que una aplicación procese las imágenes antes de proporcionarlas este como origen de imágenes personalizado. Un filtro de cámara de terceros puede procesar las imágenes de la cámara, aplicar un efecto de filtro, etc.	✓	✓	
Miniatura	Imagen de tamaño reducido tomada de una transmisión. De forma predeterminada, las miniaturas se generan cada 60 segundos, pero se puede configurar un intervalo más corto. La resolución de las miniaturas depende del tipo de canal. Consulte Grabación de contenidos en Grabación automática en Amazon S3 (streaming de baja latencia).	✓		
Metadatos cronometrados	Metadatos vinculados a marcas de tiempo específicas en una transmisión. Se pueden agregar mediante programación con la API de IVS y se asocian a fotogramas específicos. Esto garantiza que todos los espectadores reciben los metadatos en el mismo punto de la transmisión. Los metadatos cronometrados se pueden utilizar para activar acciones en el cliente, como actualiza r las estadísticas del equipo durante un evento deportivo. Consulte Incorporación de metadatos en una transmisión de video.	✓		
Transcodificación	Convierte video y audio de un formato a otro. Una transmisión entrante puede ser transcodificada a un formato diferente con múltiples velocidades de bits y resoluciones, para admitir una variedad de dispositivos de reproducción y condiciones de red.	✓	✓	

Plazo	Descripción	LL	RT	Chat
Transmuxing	Un simple reempaquetado de una transmisión ingerida a Amazon IVS, sin volver a codificar la transmisión de video. “Transmux” es la abreviatura en inglés de multiplexación de transcodificación, un proceso que cambia el formato de un archivo de audio o video manteniendo algunas o todas las transmisiones originales. Transmuxing se convierte a un formato de contenedor diferente sin cambiar el contenido del archivo. Se distingue de la transcodificación.	✓	✓	
Transmisiones variantes	Conjunto de codificaciones de la misma transmisión en distintos niveles de calidad. Cada transmisión variante se codifica como una lista de reproducción HLS independiente. Un índice de las transmisiones variantes disponibles se denomina lista de reproducción multivariante. Una vez que el reproductor de IVS recibe una lista de reproducción multivariante de IVS, puede elegir entre las transmisiones variantes durante la reproducción y cambiar sin problemas entre ellas a según cambien las condiciones de la red.	✓		
VBR	Velocidad de bits variable (VBR): un método de control de velocidad para codificadores que utiliza una velocidad de bits dinámica que cambia durante la reproducción en función del nivel de detalle necesario. Recomendamos encarecidamente no utilizar VBR debido a problemas con la calidad de video; en su lugar, utilice CBR.	✓	✓	

Plazo	Descripción	LL	RT	Chat
Vista	Una sesión de visualización única que descarga o reproduce videos de forma activa. Las vistas son la base de la cuota de vistas simultáneas. Una vista se inicia cuando una sesión de visualización comienza la reproducción de video. Una vista finaliza cuando una sesión de visualización detiene la reproducción de video. La reproducción es el único indicador de la audiencia; no se tienen en cuenta las heurísticas de interacción, como los niveles de audio, el enfoque de la pestaña del navegador y la calidad del video. Al contar las vistas, IVS no tiene en cuenta la legitimidad de los espectadores individuales ni intenta desduplicar la audiencia localizada, como varios reproductores de video en un solo equipo. Consulte Otras cuotas en Service Quotas (streaming de baja latencia).	✓		
Lector	Una persona que recibe una transmisión de IVS.	✓		
WebRTC	Web Real-Time Communication: un proyecto de código abierto que proporciona comunicación en tiempo real a los navegadores web y aplicaciones móviles. Permite que la comunicación de audio y vídeo funcione dentro de las páginas web al permitir la peer-to-peer comunicación directa, lo que elimina la necesidad de instalar complementos o descargar aplicaciones nativas. Las tecnologías detrás de WebRTC se implement an como un estándar web abierto y están disponibl es como API JavaScript normales en los principal es navegadores o como bibliotecas para clientes nativos, como Android e iOS.	✓	✓	

Plazo	Descripción	LL	RT	Chat
LÁTIGO	Protocolo de ingestión WebRTC-HTTP, un protocolo basado en HTTP que permite la ingesta de contenido basada en WebRTC en servicios de streaming y/o CDN. WHIP es un borrador del IETF desarrollado para estandarizar la ingestión de WebRTC. WHIP permite la compatibilidad con software como OBS y ofrece una alternativa (al SDK de transmisión del IVS) para la autoedición. Es posible que los streamers más sofisticados que estén familiarizados con OBS lo prefieran por sus funciones de producción avanzadas, como las transiciones de escena, la mezcla de audio y la superposición de gráficos WHIP también resulta útil en situaciones en las que no es posible o preferible utilizar el SDK para retransmisiones de IVS. Por ejemplo, en configuraciones que incluyen codificadores de hardware, el SDK de transmisión de IVS podría no ser una opción. Sin embargo, si el codificador es compatible con WHIP, puede seguir publicando directamente desde el codificador en IVS. Consulte OBS and WHIP Support.		✓	
WSS	WebSocket Seguro, un protocolo para establecer una WebSockets conexión TLS cifrada. Se utiliza para conectarse a los puntos de conexión de Chat de IVS. Consulte Step 4: Send and Receive Your First Message en Getting Started with IVS Chat.			✓

Historial del documento (transmisión en tiempo real)

Cambios en la Guía del usuario de transmisión en tiempo real

Cambio	Descripción	Fecha
Soporte para OBS y WHIP	Se agregó una nueva página. Este documento explica cómo utilizar codificadores compatibles con WHIP, como OBS, para publicar en IVS en streaming en tiempo real. WHIP (Protocolo de ingestión WebRTC-HTTP) es un borrador del IETF desarrollado para estandarizar la ingestión de WebRTC.	6 de febrero de 2024
SDK de transmisión: Android 1.14.1, iOS 1.14.1, Web 1.8.0	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en las guías del SDK de real-time-streaming transmisión: Android, iOS y Web. En la página de destino de la documentación de Amazon IVS, se actualizaron los enlaces de referencia del SDK de transmisión para señalar la nueva versión. Consulte también las Notas de la versión de Amazon IVS para esta versión. En la Guía de Android, hemos añadido un nuevo problema	1 de febrero de 2024

	conocido (un tamaño de vídeo inferior a 176 x 176). Para la guía web, hemos añadido un nuevo problema conocido. La solución alternativa consiste en limitar la resolución de vídeo a 720p al invocar o. `getUserMedia` `getDisplayMedia` En la sección Optimizaciones de transmisión en tiempo real, actualizamos la configuración de la codificación por capas con transmisión simultánea; ahora está deshabilitada de forma predeterminada.	
SDK de transmisión: Android 1.13.4, iOS 1.13.4, Web 1.7.0	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en las guías del SDK de real-time-streaming transmisión: Android, iOS y Web. En la página de destino de la documentación de Amazon IVS, se actualizaron los enlaces de referencia del SDK de transmisión para señalar la nueva versión. Consulte también las Notas de la versión de Amazon IVS para esta versión.	3 de enero de 2024

Glosario de IVS	Se amplió el glosario para incluir los términos de transmisiones en tiempo real o de baja latencia y chat de IVS.	20 de diciembre de 2023
Stage Health: nuevas CloudWatch métricas	Se cambió el nombre de la métrica PacketLoss (Stage) a DownloadPacketLoss (Stage) y se publicaron CloudWatch métricas adicionales para la transmisión en tiempo real de IVS: • DownloadPacketLoss (Escenario, participante) • DroppedFrames (Escenario, participante) • SubscribeBitrate (Escenario, participante,MediaType) Consulte Monitoreo de transmisión en tiempo real de IVS.	7 de diciembre de 2023

Políticas administradas de IAM	Se agregaron dos políticas administradas, IVS ReadOnlyAccess e FullAccess IVS. Consulte: • La nueva sección sobre políticas administradas para Amazon IVS en la página Seguridad. • Cambios en Paso 3: configurar permisos de IAM en Introducción al streaming de baja latencia de IVS.	5 de diciembre de 2023
SDK de transmisión: Android 1.13.2 e iOS 1.13.2	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en las guías del SDK de real-time-streaming transmisión: Android e iOS. En la página de destino de la documentación de Amazon IVS, se actualizaron los enlaces de referencia del SDK de transmisión para señalar la nueva versión. Consulte también las Notas de la versión de Amazon IVS para esta versión.	4 de diciembre de 2023

SDK de transmisión: Android 1.13.1	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en la guía del SDK para real-time-streaming retransmisiones: Android. En la página de destino de la documentación de Amazon IVS, se actualizaron los enlaces de referencia del SDK de transmisión para señalar la nueva versión. Consulte también las Notas de la versión de Amazon IVS para esta versión.	21 de noviembre de 2023
Service Quotas	Cambiamos la “Resolución de publicación de los participantes” de 1080p a 720p.	18 de noviembre de 2023

SDK de transmisión: Android 1.13.0 e iOS 1.13.0	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en las guías del SDK de real-time-streaming transmisión: Android e iOS. En la página de destino de la documentación de Amazon IVS, se actualizaron los enlaces de referencia del SDK de transmisión para señalar la nueva versión. Consulte también las Notas de la versión de Amazon IVS para esta versión. También hemos realizado varias actualizaciones en las Optimizaciones de transmisión. Entre otras cosas, la característica “Transmisión adaptativa: codificación por capas con transmisión simultánea” ahora requiere una suscripción explícita y solo es compatible con las versiones recientes del SDK.	17 de noviembre de 2023

Grabación compuesta	Se realizaron los siguientes cambios: • Se agregó una página de Grabación compuesta para esta nueva característica. • Se actualizó la política de Introducción a transmisión en tiempo real de IVS con puntos de conexión S3 en la sección “Configurar los permisos de IAM”. • Service Quotas actualiza das con cuotas de tarifas de llamada para los nuevos puntos de conexión.	16 de noviembre de 2023

Composición del servidor (SSC)	La composición del servidor de IVS permite a los clientes transferir la composición y la transmisión de un escenario de IVS a un servicio gestionado por IVS. Las transmisiones SSC y RTMP a un canal se invocan a través de los puntos de conexión del plano de control del IVS en la región de origen del escenario. Consulte: • Primeros pasos: en la sección “Configurar los permisos de IAM”, añadimos los puntos de conexión SSC a la política. • Uso de Amazon EventBridge con IVS: hemos añadido nuevas métricas. • Composición del servidor: este nuevo documento incluye una descripción general e instrucciones de configuración. • Service Quotas: agregamos nuevos límites de tarifas de llamadas y otras cuotas. Consulte también: • Cambios enumerados a continuación en Cambios de referencia de la API de	16 de noviembre de 2023

	transmisión en tiempo real de IVS. • Cambios enumerados en el Historial del documento (transmisión de baja latencia).	
SDK de transmisión de IVS	En la Descripción general del SDK de transmisión, actualizamos los Requisitos de plataforma > Plataformas nativas para aclarar qué versiones del SDK son compatibles y añadimos “Navegadores móviles (iOS y Android)”. En la Guía web de transmisión, añadimos “Limitaciones de la web móvil”.	9 de noviembre de 2023
SDK de transmisión de IVS	Hemos añadido una nueva página sobre Filtros de cámara de terceros.	9 de noviembre de 2023
Introducción a transmisión en tiempo real de IVS	Hemos actualizado los procedimientos en Configurar permisos de IAM.	20 de octubre de 2023
Monitoreo de transmisión en tiempo real	En CloudWatch Metrics: IVS Real-Time Streaming, añadimos valores de muestra para las dimensiones.	17 de octubre de 2023

SDK de transmisión: guía para la web	Hemos realizado varios cambios en el Monitoreo del estado de sonido desactivado en los contenidos multimedia del participante remoto.	17 de octubre de 2023
SDK de transmisión: web 1.6.0	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en la guía del SDK de real-time-streaming difusión: Web. La página de inicio de la documentación de Amazon IVS siempre indica la versión reciente de las Referencias del SDK del Transmisor. Consulte también las Notas de la versión de Amazon IVS para esta versión. En la Guía Web, en la sección «Recuperar un MediaStream archivo de un dispositivo», también eliminamos las dos `max` líneas; la mejor práctica es especificarlas únicament `eideal`. En Optimizaciones de transmisión en tiempo real, agregamos una nueva sección, Optimización de la tasa de bits de audio y soporte estéreo.	16 de octubre de 2023

Stage Health: nuevas CloudWatch métricas	Se publicaron CloudWatch métricas para la transmisión en tiempo real de IVS. Consulte Monitoreo de transmisión en tiempo real de IVS.	12 de octubre de 2023
SDK de transmisión: Android 1.12.1	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en la guía del SDK para real-time-streaming retransmisiones: Android. También se agregó una nueva sección, Uso de micrófonos Bluetooth. La página de inicio de la documentación de Amazon IVS siempre indica la versión reciente de las Referencias del SDK del Transmisor. Consulte también las Notas de la versión de Amazon IVS para esta versión.	12 de octubre de 2023

SDK de transmisión: web 1.5.2	El número de versión actualizado y los enlaces a los artefactos de la nueva versión se encuentran en la guía del SDK para real-time-streaming retransmisiones: Web. La página de inicio de la documentación de Amazon IVS siempre indica la versión reciente de las Referencias del SDK del Transmisor. Consulte también las Notas de la versión de Amazon IVS para esta versión.	14 de septiembre de 2023
Introducción a transmisión en tiempo real de IVS	En Android > Instalar el SDK de transmisión, se agregó el enlace de datos.	12 de septiembre de 2023
Control de errores del SDK de transmisión	Se agregaron secciones de “Gestión de errores” a las guías del SDK de transmisión: Web, Android e iOS.	12 de septiembre de 2023
Introducción a transmisión en tiempo real de IVS	En Distribuir los tokens de participante, se agregó una nota Importante sobre no crear funciones basadas en el formato de token actual.	1 de septiembre de 2023
Introducción a transmisión en tiempo real de IVS	En Configurar los permisos de IAM, se actualizó el conjunto de permisos.	31 de agosto de 2023

SDK de transmisión: Web 1.5.1, Android 1.12.0 y iOS 1.12.0	Se actualizaron el número de versión y los enlaces a los artefactos de la nueva versión, en las guías del SDK de real-time-streaming difusión: Web, Android e iOS. En la página de destino de la documentación de Amazon IVS, se actualizaron los enlaces de referencia del SDK de transmisión para señalar la nueva versión. Consulte también las Notas de la versión de Amazon IVS para esta versión.	23 de agosto de 2023

Lanzamiento de transmisión en tiempo real	Esta versión incluye cambios importantes en la documentación. Cambiamos el nombre de la documentación anterior por transmisión en tiempo real de baja latencia de IVS y publicamos la nueva documentación de transmisión en tiempo real de IVS. La página de destino de la documentación de Amazon IVS ahora tiene secciones diferentes para la transmisión en tiempo real y la transmisión de baja latencia. Cada sección tiene su propia guía del usuario y referencia de API. Para ver otros cambios en la documentación, consulte Historial del documento (transmisión de baja latencia).	7 de agosto de 2023

SDK de transmisión: Web 1.5.0, Android 1.11.0 y iOS 1.11.0	Se actualizaron el número de versión y los enlaces de los artefactos para la nueva versión en las guías del SDK de transmisión: Web, Android e iOS. En la página de destino de la documentación de Amazon IVS, se actualizaron los enlaces de referencia del SDK de transmisión para señalar la nueva versión. Consulte también las Notas de la versión de Amazon IVS para esta versión.	7 de agosto de 2023

Cambios en la Referencia de la API de transmisión en tiempo real de IVS

Cambio de la API	Descripción	Fecha
Grabación compuesta	Hemos añadido 4 StorageConfiguration puntos finales y 7 objetos (DestinationDetail,, S3 Recording Configuration, S3DetailDestinationConfiguration, S3StorageConfiguration,,). StorageConfiguration StorageConfigurationSummary Modificamos 3 objetos (composición, destino). DestinationConfiguration Esto afecta a la GetCompos ition respuesta y a la StartComposition solicitud y la respuesta.	16 de noviembre de 2023

Cambio de la API	Descripción	Fecha
Composición del servidor	Hemos añadido 8 elementos de composición y EncoderConfiguration puntos finales y 11 objetos (composición ChannelDestinationConfiguration CompositionSummary, destino DestinationConfiguration, DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration LayoutConfiguration, y vídeo).	16 de noviembre de 2023
Estado de los escenarios: datos sobre nuevos participantes	Se agregaron seis campos al objeto participante: `browserName` , `browserVersion` , `ispName`, `osName`, `osVersion` y `sdkVersion` . Esto afecta a la GetParticipant respuesta.	12 de octubre de 2023
Token de participante	Se agregó una nota Importante sobre no crear funciones basadas en el formato de token actual.	1 de septiembre de 2023
Lanzamiento de transmisión en tiempo real de IVS	Esta versión incluye cambios importantes en la documentación. Cambiamos el nombre de la documentación anterior por transmisión en tiempo real de baja latencia de IVS y publicamos la nueva documentación de transmisión en tiempo real de IVS. La página de destino de la documentación de Amazon IVS ahora tiene secciones diferentes para la transmisión en tiempo real y la transmisión de baja latencia. Cada sección tiene su propia guía del usuario y referencia de API. La Referencia de la API de transmisión en tiempo real de IVS forma parte de la documentación de transmisión en tiempo real de IVS. Anteriormente, se titulaba Referencia de la API de escenarios de IVS. Su historial anterior se describe en Historial del documento (transmisión de baja latencia).	7 de agosto de 2023

Notas de la versión (transmisión en tiempo real)

6 de febrero de 2024

Soporte para OBS y WHIP

El IVS se puede utilizar con codificadores compatibles con WHIP, como OBS, para publicar en IVS en streaming en tiempo real. WHIP (Protocolo de ingestión WebRTC-HTTP) es un borrador del IETF desarrollado para estandarizar la ingestión de WebRTC. Consulte la nueva página sobre OBS y WHIP Support.

1 de febrero de 2024

Amazon IVS Broadcast SDK: Android 1.14.1, iOS 1.14.1, Web 1.8.0 (transmisión en tiempo real)

Plataforma	Descargas y cambios
SDK de transmisión web 1.8.0	Documentación de referencia: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • La codificación por capas con transmisión simultánea ahora está deshabilitada de forma predeterminada. • Se ha corregido un error que provocaba que una instancia de un escenario no se desconectara correctamente cuando se eliminaba un escenario o cuando un participante se desconectaba del servidor. El SDK ahora emite un `STAGE_CONNECTION_STATE_CHANGED` evento con un estado de `DISCONNECTED` (en lugar de `ERRORED` y luego`CONNECTING`).

Plataforma	Descargas y cambios
	• Se ha corregido un problema por el que la publicación fallaba al actualizar la estrategia con pistas de audio o vídeo vacías.
SDK de transmisión para Android 1.14.1	Documentación de referencia: https://aws.github.io/ /1.14.1/android amazon-ivs-broadcast-docs • La codificación por capas con transmisión simultánea ahora está deshabilitada de forma predeterminada. • Se ha actualizado `libWebRTC` del M108 al M119. • Se corrigieron varios bloqueos para mejorar la estabilidad general. • Se ha añadido soporte para la publicación en estéreo. Esto se puede activar a través del `StageAudioConfiguration` objeto. • Se ha corregido un error que provocaba que el feed de los participantes quedara en negro después de unirse a una sesión. • Se han actualizado `libWebRTC` las referencias internas para evitar conflictos de símbolos cuando se incluyen otras `libWebRTC` versiones en la misma aplicación host.

Plataforma	Descargas y cambios
SDK de transmisión para iOS 1.14.1	Descárguelo para su transmisión en tiempo real: https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip Documentación de referencia: https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios • La codificación por capas con transmisión simultánea ahora está deshabilitada de forma predeterminada. • Se ha actualizado `libWebRTC` del M108 al M119. • Se corrigieron varios bloqueos para mejorar la estabilidad general. • Se ha añadido soporte para la publicación en estéreo. Esto se puede habilitar mediante`IVSLocalStageStreamAudioConfiguration`. • Se corrigió un bloqueo al habilitar el modo de solo audio para otros participantes. • Se mejoró el TTV y se redujo el tamaño de los binarios.

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5,223 MB	13.118 MB
armeabi-v7a	4.524 MB	9.134 MB
x86_64	5.418 MB	13.955 MB

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
x86	5.61 MB	14.369 MB

Tamaño del SDK de transmisión: iOS

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64	3.350 MB	7.790 MB

3 de enero de 2024

Amazon IVS Broadcast SDK: Android 1.13.4, iOS 1.13.4, Web 1.7.0 (transmisión en tiempo real)

Plataforma	Descargas y cambios
SDK de transmisión web 1.7.0	Documentación de referencia: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • Mejorado time-to-video para los suscriptores que se unen a las etapas. • Eliminó la `minAudioBitrateKbps` propiedad (no estaba utilizada). • Se ha mejorado la recuperación de la red durante cortes o cambios de Internet.
SDK de Android Broadcast 1.13.4	Documentación de referencia: https://aws.github.io/ /1.13.4/android amazon-ivs-broadcast-docs • StageAudioConfiguration ahora permite configurar si la cancelación de eco debe estar habilitada.

Plataforma	Descargas y cambios
SDK de transmisión para iOS 1.13.4	Descarga para streaming en tiempo real: https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip Documentación de referencia: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.4/ios • En iOS, hemos mejorado el motor de audio tanto para la grabación como para la reproducción, centrándonos en la estabilidad y la capacidad de recuperación. Esto mejora la compatibilidad con los cambios de ruta durante el uso, mejora la recuperación de la batería en los casos extremos y reduce la cantidad de bloqueos del hilo principal. • Se ha corregido un error que provocaba que el micrófono permaneciera activo incluso después de haberlo separado de un escenario, dejando encendido el indicador de privacidad de iOS. (El SDK no procesaba el audio entrante en ese momento).

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5,187 MB	13.025 MB
armeabi-v7a	4.491 MB	9.056 MB
x86_64	5.359 MB	13.829 MB
x86	5.553 MB	14.214 MB

Tamaño del SDK de transmisión: iOS

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64	3,45 MB	7,84 MB

7 de diciembre de 2023

Nuevas métricas CloudWatch

Cambiamos el nombre de la métrica PacketLoss (Etapa) a DownloadPacketLoss (Etapa). También publicamos CloudWatch métricas adicionales para la transmisión en tiempo real de IVS:

- DownloadPacketLoss (Escenario, participante)
- DroppedFrames (Escenario, participante)
- SubscribeBitrate (Escenario, participante,MediaType)

Para obtener más información, consulte Monitoreo de transmisión en tiempo real de IVS.

4 de diciembre de 2023

SDK de transmisión de Amazon IVS: Android 1.13.2 e iOS 1.13.2 (transmisión en tiempo real)

Plataforma	Descargas y cambios
Todos los dispositivos móviles (Android e iOS)	• La configuración de supresión de ruido está disponible para que los desarrolladores la habiliten o deshabiliten para su publicación.
SDK de transmisión para Android 1.13.2	Documentación de referencia: https:// aws.github.io/ amazon-ivs-broadcast- docs /1.13.2/android

Plataforma	Descargas y cambios
	• Se ha mejorado el tiempo que se tarda en cargar el video (TTV) al unirse al primer escenario de una sesión.
SDK de transmisión para iOS 1.13.2	Descarga para streaming en tiempo real: https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip Documentación de referencia: https:// aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/ios • No hay cambios en el SDK en tiempo real.

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5177 MB	13,01 MB
armeabi-v7a	4485 MB	9,045 MB
x86_64	5,352 MB	13,808 MB
x86	5547 MB	14 192 MB

Tamaño del SDK de transmisión: iOS

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64	3,45 MB	7,82 MB

21 de noviembre de 2023

SDK de transmisión de Amazon IVS: Android 1.13.1 (transmisión en tiempo real)

Plataforma	Descargas y cambios
SDK de transmisión para Android 1.13.1	Documentación de referencia: https://aws.github.io/ /1.13.1/android amazon-ivs-broadcast-docs • Se ha corregido un error que provocaba un bloqueo al salir, soltar y volver a entrar rápidamente en el mismo escenario.

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5177 MB	13 102 MB
armeabi-v7a	4485 MB	9046 MB
x86_64	5353 MB	13 809 MB
x86	5547 MB	14 192 MB

17 de noviembre de 2023

SDK de transmisión de Amazon IVS: Android 1.13.0 e iOS 1.13.0 (transmisión en tiempo real)

Plataforma	Descargas y cambios
Todos los dispositivos móviles (Android e iOS)	• Optimizaciones de transmisión actualizado. Entre otras cosas, la característica “Transmisión adaptativa: codificación por capas con transmisión simultánea” ahora requiere una suscripción explícita y solo es compatible con las versiones recientes de SDK. • Se mejoró la estabilidad de las fases al reducir los fallos poco frecuentes. • Se ha mejorado el tiempo que se tarda en cargar el vídeo (TTV) al unirse a un escenario. • Se ha mejorado la experiencia con los dispositivos Bluetooth. • Se optimizó el uso de la CPU y la memoria del SDK y se redujo el tamaño de la biblioteca. • Se agregó la clase `StageAudioManager` , que se puede usar para configurar los parámetros de captura y reproducción de audio, incluidos los ajustes preestablecidos para la comunicación de voz, la reproducción multimedia y más. Para obtener más información, consulte la nueva página, IVS Broadcast SDK: Mobile Audio Modes. • Se agregó una nueva función `requestQualityStats` para mostrar eventos de calidad estructurados a partir de las estadísticas de WebRTC.

Plataforma	Descargas y cambios
	• Se agregó una nueva función para actualizar la tasa de bits del audio. Se establece en `LocalStageStream` objetos igual que en la configuración de vídeo, pero a través de un nuevo objeto de configuración de audio.

Plataforma	Descargas y cambios

SDK de transmisión para Android 1.13.0	Documentación de referencia: https://aws.github.io/ /1.13.0/android amazon-ivs-broadcast-docs
	
	• Todos los métodos de la interfaz `StageRenderer` ahora son opcionales.
	• Se ha añadido compatibilidad de `SurfaceView` con la vista previa basada en datos para mejorar el rendimiento. Los métodos existentes `getPreview` en `Session` y `StageStream` siguen devolviendo una subclase de `TextureView`, pero esto puede cambiar en una futura versión del SDK.
	  • Si su aplicación depende específicamente de `TextureView`, puede continuar sin cambios. También puede cambiar de `getPreview` a `getPreviewTextureView` para prepararse para el eventual cambio de lo que `getPreview` devuelve como valor predeterminado.
	  • Si su aplicación no requiere `TextureView` específicamente, le recomendamos que cambie a `getPreviewSurfaceView` para reducir el uso de CPU y memoria.
	• El SDK ahora implementa un nuevo tipo de vista previa denominada `ImagePreviewSurfaceTarget` que funciona con el objeto Android Surface proporcionado por la aplicación. No es una subclase de Android View, que proporciona una mayor flexibilidad.
	• Se corrigió el caso en el que la devolución de llamada `onFrame` para un participante

Plataforma	Descargas y cambios
	remoto se llamaba en el momento incorrecto y con un tamaño incorrecto. • `SurfaceSource # getInputSurface` ahora está anotado con `@Nullable` . Debe comprobar su código antes de usarlo. • Se han añadido `UserId`, `attributes` y a `ParticipantInfo` . Las propiedades `UserId` y `attributes` están integrada s en el token y las aplicaciones pueden recuperarlas a través de `Participa ntInfo` siempre que un participante se una. • La captura de cámara y el procesamiento de la vista previa ahora tienen un valor predeterminado de 720 x 1280 o la resolució n de publicación (la que sea mayor) es de 15 fps. Puede ajustar la resolución o los fps utilizando `StageVideoConfiguration # setCameraCaptureQuality` . • `IllegalArgumentException` que aparece al establecer las propiedades de configuración ahora incluye el valor proporcio nado en el mensaje de excepción.

Plataforma	Descargas y cambios
SDK de transmisión para iOS 1.13.0	Descarga para transmisión en tiempo real: https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentación de referencia: https://aws.github.io/ /1.13.0/ios amazon-ivs-broadcast-docs • Se solucionó el problema por el que el SDK no cambiaba la configuración de vídeo si la configuración de vídeo se actualizaba antes de la publicación. • Se incorporó la solución de Google para una vulnerabilidad de seguridad de LibVPX (CVE-2023-5217). (Tenga en cuenta que el SDK de Android no requirió ningún cambio por este problema). • Las aplicaciones que utilicen otras bibliotecas que incluyan `libWebRTC` ya no tendrán conflictos con el SDK de IVS Broadcast. • Todos los métodos del protocolo `IVSStageRenderer` ahora están marcados como `@optional` . • Los micrófonos y las cámaras devueltos por nuestros SDK ahora tienen un orden de clasificación garantizado, tal y como se documenta en los propios SDK. • Ahora, varias cámaras pueden tener un valor de `true` para su propiedad de `isDefault` , uno para cada posición, según lo determine el sistema operativo. • Se ha añadido `IVSStageAudioManager` , que permite un control preciso de la función subyacente `AVAudioSession` para

Plataforma	Descargas y cambios
	ampliar la variedad de casos de uso de la funcionalidad de Escenarios. • Se agregó `UserId` a `ParticipantInfo` .

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5,17 MB	13,00 MB
armeabi-v7a	4,48 MB	9,04 MB
x86_64	5,35 MB	13,80 MB
x86	5,54 MB	14,18 MB

Tamaño del SDK de transmisión: iOS

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64	3,45 MB	7,84 MB

16 de noviembre de 2023

Grabación compuesta

Esta nueva característica permite grabar la vista compuesta de un IVS Stage en un bucket de Amazon S3. Para obtener más información, consulte:

- Grabación compuesta: esta es una página nueva.
- Introducción a transmisión de IVS en tiempo real: en la sección “Configurar los permisos de IAM”, añadimos los puntos de conexión S3 a la política.

- Service Quotas de IVS: se agregaron cuotas de tarifas de llamada para los nuevos puntos de conexión.
- Referencia de la API de transmisión en tiempo real de IVS: agregamos 4 StorageConfiguration puntos finales y 7 objetos (DestinationDetail,, S3, RecordingConfiguration S3Detail, S3DestinationConfiguration,). StorageConfiguration StorageConfiguration StorageConfigurationSummary También modificamos 3 objetos (composición, destino DestinationConfiguration); esto afecta a la GetComposition respuesta y a la solicitud y la StartComposition respuesta.

16 de noviembre de 2023

Composición del servidor

La composición del servidor de IVS permite a los clientes transferir la composición y la emisión de un escenario de IVS a un servicio gestionado por IVS. La composición del servidor y la transmisión RTMP a un canal se invocan a través de los puntos de conexión del plano de control del IVS en la región de origen del escenario. Para obtener más información, consulte:

- Introducción a transmisión de IVS en tiempo real: en el apartado “Configurar los permisos de IAM”, añadimos puntos de conexión de SSC a la política.
- Uso de Amazon EventBridge con IVS Real-Time Streaming: hemos añadido nuevas métricas.
- Composición del servidor: este nuevo documento incluye una descripción general e instrucciones de configuración.
- Service Quotas (transmisión en tiempo real): agregamos nuevos límites de tarifa de llamadas y otras cuotas.
- Referencia de la API de streaming en tiempo real: hemos añadido 8 componentes y EncoderConfiguration puntos finales y 11 objetos (composición ChannelDestinationConfiguration CompositionSummary, destino DestinationConfiguration, DestinationSummary, EncoderConfiguration,, EncoderConfigurationSummary GridConfiguration LayoutConfiguration, y vídeo).

En la Guía del usuario de transmisión de baja latencia de IVS, consulte:

- Habilitación de varios hosts en una transmisión de IVS: agregamos “Transmisión de un escenario: composición del cliente versus composición del servidor” y actualizamos “4. Transmisión del escenario”.

16 de octubre de 2023

SDK de transmisión de Amazon IVS: Web 1.6.0 (transmisión en tiempo real)

Plataforma	Descargas y cambios
SDK de transmisión web 1.6.0	Documentación de referencia: https://aws.github.io/ amazon-ivs-web-broadcast /docs/ sdk-reference • Tiempo de reproducción de video (TTV) mejorado. • Configuración adicional `maxAudioBitrate` , que admite hasta 128 kbps de canales de audio mono o estéreo.

12 de octubre de 2023

Nuevas métricas y datos de los participantes CloudWatch

Publicamos CloudWatch las métricas para la transmisión en tiempo real de IVS. Para obtener más información, consulte Monitoreo de transmisión en tiempo real de IVS.

También hemos agregado seis campos al objeto de la API de participantes: `browserName`, `browserVersion`, `ispName`, `osName`, `osVersion` y `sdkVersion`. Esto afecta a la GetParticipant respuesta. Consulte la Referencia de la API de transmisión en tiempo real de IVS.

12 de octubre de 2023

SDK de transmisión de Amazon IVS: Android 1.12.1 (transmisión en tiempo real)

Plataforma	Descargas y cambios
SDK de transmisión para Android 1.12.1	Documentación de referencia: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.1/android • Se ha corregido un error por el cual `BroadcastSession.setListener` provocaba un error al llamar.

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5,853 MB	16,375 MB
armeabi-v7a	4,895 MB	10,803 MB
x86_64	6,149 MB	17,318 MB
x86	6,328 MB	17,186 MB

14 de septiembre de 2023

SDK de transmisión de Amazon IVS: Web 1.5.2 (transmisión en tiempo real)

Plataforma	Descargas y cambios
SDK de transmisión para web 1.5.2	Documentación de referencia: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • Se ha corregido un error que impedía volver a publicar `refreshStrategy` cuando el estado publicado entraba en un estado `ERRORED`.

23 de agosto de 2023

SDK de transmisión de Amazon IVS: Web 1.5.1, Android 1.12.0 e iOS 1.12.0 (transmisión en tiempo real)

Plataforma	Descargas y cambios
SDK de transmisión para web 1.5.1	Documentación de referencia: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • Se ha corregido un error con los tipos internos de Maybe en la versión 5. TypeScript • Se ha agregado una mejor detección para la compatibilidad con la transmisión simultánea. • Se han corregido dos condiciones de carrera con `refreshStrategy` al intentar publicar.

Plataforma	Descargas y cambios
	• Se ha corregido una condición de carrera con `refreshStrategy` al intentar actualizar a los participantes a los que suscribirse.
Todos los dispositivos móviles (Android e iOS)	• Se ha solucionado un problema poco frecuente por el que la acción de publicación nunca se completaba. • Se mejoró la estabilidad de las fases al reducir los fallos poco frecuentes. • Se ha mejorado la estabilidad de las etapas al resolver los problemas relacionados con las condiciones de la carrera que se producían al entrar o salir rápidamente. • Se ha agregado un nuevo método `setOnFrameCallback`, en `ImageDevice`. Esto permite la observación a medida que los fotogramas se transfieren al propio dispositivo, lo que permite obtener información sobre la relación de aspecto de las imágenes más recientes. Este método también se puede utilizar para detectar cuándo se renderiza el primer fotograma para un participante remoto en un escenario.
SDK de transmisión para Android 1.12.0	Documentación de referencia: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.0/android • Ahora se admite Android 9. • Uso y rendimiento mejorados de la CPU.

Plataforma	Descargas y cambios
SDK de transmisión para iOS 1.12.0	Descarga para transmisión en tiempo real: https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentación de referencia: https://aws.github.io/ /1.12.0/ios amazon-ivs-broadcast-docs • Se corrigió la firma de `IVSDeviceDiscovery.createAudioSourceWithName` para devolver un `IVSCustomAudioSource` en lugar de `IVSCustomImageSource`.

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5,853 MB	16,375 MB
armeabi-v7a	4,895 MB	10,803 MB
x86_64	6,149 MB	17,318 MB
x86	6,328 MB	17,186 MB

Tamaño del SDK de transmisión: iOS

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64	5,06 MB	10,92 MB

7 de agosto de 2023

SDK de transmisión de Amazon IVS: Web 1.5.0, Android 1.11.0 y iOS 1.11.0

Plataforma	Descargas y cambios
SDK de transmisión para Web 1.5.0	Documentación de referencia: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • Transmisión simultánea agregada: cuando está habilitada, esta característica permite al publicador enviar capas de video de alta y baja calidad. Los suscriptores seleccionan automáticamente la calidad óptima en función de las condiciones de la red. Consulte Optimización del contenido multimedia.
Todos los dispositivos móviles (Android e iOS)	Transmisión simultánea agregada: cuando está habilitada, esta característica permite al publicador enviar capas de video de alta y baja calidad. Los suscriptores seleccionan automáticamente la calidad óptima en función de las condiciones de la red. Consulte “Activación o desactivación de la codificación en capas con transmisión simultánea” en las Guías del SDK de transmisión para Android y iOS.
SDK de transmisión para Android 1.11.0	Documentación de referencia: https://aws.github.io/ /1.11.0/android amazon-ivs-broadcast-docs • Se ha corregido un error que provocaba que la creación de muchos escenarios acabara

Plataforma	Descargas y cambios
	causando un bloqueo. (El número exacto de escenarios depende del dispositivo).
SDK de transmisión para iOS 1.11.0	Descarga para transmisión en tiempo real: https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentación de referencia: https://aws.github.io/ /1.11.0/ios amazon-ivs-broadcast-docs • Se corrigió la firma de `IVSDeviceDiscovery.createAudioSourceWithName` para devolver `IVSCustomAudioSource` en lugar de `IVSCustomImageSource`.

Tamaño del SDK de transmisión: Android

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64-v8a	5,811 MB	16,186 MB
armeabi-v7a	4,857 MB	10,646 MB
x86_64	6,108 MB	17,122 MB
x86	6,289 MB	16,994 MB

Tamaño del SDK de transmisión: iOS

Arquitectura	Tamaño comprimido	Tamaño sin comprimir
arm64	5,030 MB	10,810 MB

7 de agosto de 2023

Transmisión en tiempo real

La transmisión en tiempo real de Amazon Interactive Video Service (IVS) le permite ofrecer transmisiones en directo con una latencia inferior a 300 milisegundos desde el host hasta el espectador.

Esta versión incluye cambios importantes en la documentación. La página de destino de la documentación de Amazon IVS ahora tiene secciones diferentes para la transmisión en tiempo real y la transmisión de baja latencia. Cada sección tiene su propia guía del usuario y referencia de API. Para obtener más información sobre el documento, consulte el historial del documento (para tiempo real y baja latencia). Para la transmisión en tiempo real, comience con la Guía del usuario de transmisión en tiempo real de IVS y la Referencia de la API de transmisión en tiempo real de IVS.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.