Guia do usuário do streaming em tempo real

# Amazon IVS

# Amazon IVS: Guia do usuário do streaming em tempo real

# Table of Contents

# O que é o Streaming em tempo real do Amazon IVS?

O Streaming em tempo real do Amazon Interactive Video Service (IVS) oferece tudo o que você precisa para adicionar áudio e vídeo em tempo real às suas aplicações.

Pontos fortes:

- Latência em tempo real: crie aplicações para casos de uso sensíveis à latência, ajudando seus espectadores a permanecerem conectados e engajados com o streaming em tempo real do IVS. Faça streams ao vivo com uma latência que pode ser inferior a 300 milissegundos do host ao espectador.
- Alta simultaneidade: desbloqueie o potencial de interações em larga escala com o streaming em tempo real do IVS. Acomode audiências de até 10 mil espectadores e possibilite que até 12 hosts subam ao palco virtual.
- Otimização para dispositivos móveis: o streaming em tempo real do IVS é otimizado para casos de uso de dispositivos móveis, atendendo a uma ampla variedade de dispositivos e funcionalidades de rede. Ao integrar os SDKs de Transmissão do Amazon IVS para Android e iOS, seus usuários podem participar como hosts ou espectadores, desfrutando de streams ao vivo de alta qualidade em seus dispositivos móveis.

Casos de uso:

- Pontos para convidados: crie aplicações que permitem que os hosts promovam convidados “no palco”, transformando espectadores em hosts para interações em tempo real.
- Modo Versus (VS): produza experiências com competições lado a lado e permita que os espectadores assistam aos hosts competirem em tempo real.
- Salas de áudio: convide receptores para participar da conversa como convidados e promova um envolvimento mais aprofundado em suas salas de áudio.
- Leilões em vídeo ao vivo: transforme os leilões em eventos de vídeo interativos e mantenha seu entusiasmo e integridade com uma latência em tempo real.

Além da documentação do produto fornecida aqui, consulte https://ivs.rocks/, um site dedicado para navegar pelo conteúdo publicado (demonstrações, amostras de código, publicações de blog), calcular o custo e experimentar o Amazon IVS com demonstrações ao vivo.

# Solução global, controle regional

## O streaming e a visualização são globais

Você pode usar o Amazon IVS para fazer streaming para visualizadores em todo o mundo:

- Quando você faz streaming, o Amazon IVS ingere automaticamente o vídeo em um local próximo a você.
- Os espectadores podem assistir seus streams ao vivo globalmente.

Outra maneira de dizer isso é que o “plano de dados” é global. O plano de dados refere-se ao streaming/ingestão e visualização.

## O controle é regional

Embora o plano de dados do Amazon IVS seja global, o “plano de controle” é regional. O ambiente de gerenciamento refere-se ao console, à API e aos recursos (palcos) do Amazon IVS.

Outra maneira de dizer isso é que o Amazon IVS é um “serviço regional da AWS”. Ou seja, os recursos do Amazon IVS em cada região são independentes de recursos semelhantes em outras regiões. Por exemplo, um palco criado em uma região é independente dos palcos criados em outras regiões.

Ao utilizar recursos (por exemplo, a criação de um palco), você deve especificar a região em que ele será criado. Posteriormente, ao gerenciar recursos, você deve fazê-lo na mesma região em que foram criados.

| Se você usar o(a)... | Especifique a região da seguinte forma... |
|---|---|
| Console do Amazon IVS | Uso do menu suspenso Select a Region (Selecione uma região) no canto superior direito da barra de navegação. |
| API do Amazon IVS | Uso do endpoint de serviço apropriado. Consulte a Referência de API do Streaming em tempo real do Amazon IVS.<br><br>(Se você acessar a API por meio de um SDK, configure o parâmetro `region` do SDK. Consulte Ferramentas para criar com a AWS.) |

| Se você usar o(a)... | Especifique a região da seguinte forma... |
| --- | --- |
| CLI do AWS | Há duas opções:<br>• Anexar `--region <aws-region>` ao seu comando da CLI.<br>• Colocar a região em seu arquivo de configuração local da AWS. |

Lembre-se de que, independentemente da região em que um palco foi criado, é possível realizar transmissões para o Amazon IVS de qualquer lugar e os espectadores podem assistir de qualquer lugar.

# Conceitos básicos do Streaming em tempo real do IVS

Este documento descreve as etapas envolvidas na integração do Streaming em tempo real do Amazon IVS com a sua aplicação.

Tópicos



# Introdução

## Pré-requisitos

Antes de usar o streaming em tempo real pela primeira vez, conclua as tarefas a seguir. Para obter instruções, consulte Conceitos básicos do streaming de baixa latência do IVS.

- Criar uma conta da AWS
- Configurar os usuários raiz e administrativo

## Outras referências

- Referência do SDK de Transmissão do IVS para Web
- Referência do SDK de Transmissão do IVS para Android
- Referência do SDK de Transmissão do IVS para iOS
- Referência de API do Streaming em tempo real do IVS

# Terminologia do streaming em tempo real

| Prazo | Descrição | |
|---|---|---|
| Estágio | Um espaço virtual no qual os participantes podem trocar vídeos em tempo real. | |
| Host | Um participante que envia vídeos locais para o palco. | |
| Visualizador | Um participante que recebe vídeos dos hosts. | |
| Participante | Um usuário conectado ao palco como host ou espectador. | |
| Token de participante | Um token que autentica um participante quando ele ingressa em um palco. | |
| SDK de transmissão | Uma biblioteca do cliente que possibilita aos participantes enviar e receber vídeos. | |

## Visão geral das etapas

1. the section called “Definir as permissões do IAM”: crie uma política do AWS Identity and Access Management (IAM) que conceda aos usuários um conjunto básico de permissões e atribua essa política aos usuários.
2. Criar um palco: crie um espaço virtual no qual os participantes possam trocar vídeos em tempo real.
3. Distribuir tokens de participantes: envie tokens aos participantes para que eles possam ingressar no seu palco.

4. Integrar o SDK de Transmissão do IVS: adicione o SDK de transmissão à sua aplicação para possibilitar que os participantes enviem e recebam vídeos: the section called “Web”, the section called “Android” e the section called “iOS”.
5. Publicar e inscrever-se em vídeos: envie seu vídeo para o palco e receba vídeos de outros hosts: the section called “Web”, the section called “Android” e the section called “iOS”.

# Definir as permissões do IAM

Em seguida, você deverá criar uma política do AWS Identity and Access Management (IAM) que conceda aos usuários um conjunto básico de permissões (por exemplo, para criar um palco do Amazon IVS e criar tokens de participante) e atribuir essa política a usuários. É possível atribuir as permissões ao criar um novo usuário ou adicionar as permissões a um usuário existente. Os dois procedimentos são apresentados abaixo.

Para obter mais informações (por exemplo, para aprender sobre usuários e políticas do IAM, como anexar uma política a um usuário e como restringir o que os usuários podem fazer com o Amazon IVS), consulte:

- Como criar um usuário do IAM no Guia do usuário do IAM
- As informações em Segurança do Amazon IVS sobre IAM e “Managed Policies for IVS”.
- As informações sobre o IAM apresentadas em Amazon IVS Security

Você pode usar uma política gerenciada pela AWS existente para o Amazon IVS ou criar uma política que personalize as permissões que você quer conceder a um conjunto de usuários, grupos ou perfis. Ambas as abordagens são descritas a seguir.

## Usar uma política existente para permissões do IVS

Na maioria dos casos, você vai querer usar uma política gerenciada pela AWS para o Amazon IVS. Elas são descritas totalmente na seção Managed Policies for IVS da Segurança do IVS.

- Use a política `IVSReadOnlyAccess` gerenciada pela AWS para dar aos desenvolvedores de aplicações acesso a todos os endpoints das APIs Get e List do IVS (para streaming de baixa latência e em tempo real).
- Use a política `IVSFullAccess` gerenciada pela AWS para dar aos desenvolvedores de aplicações acesso a todos os endpoints da API do IVS (para streaming de baixa latência e em tempo real).

# Opcional: criar uma política personalizada para permissões do Amazon IVS

Siga estas etapas:

1. Faça login no Console de Gerenciamento da AWS e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. No painel de navegação, selecione Políticas e, em seguida, Criar política. Uma janela Especificar permissões é aberta.
3. Na janela Especificar permissões, escolha a guia JSON, e copie e cole a política do IVS a seguir na área de texto do Editor de políticas. (A política não inclui todas as ações do Amazon IVS. Você pode adicionar/excluir [Permitir/Negar] permissões de acesso ao endpoint, conforme necessário. Consulte IVS Real-Time Streaming API Reference para obter detalhes sobre endpoints do 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. Ainda na janela Especificar permissões, escolha Avançar (role até a parte inferior da janela para ver isso). Uma janela Revisar e criar é aberta.
5. Na janela Revisar e criar, insira um Nome da política e, opcionalmente, adicione uma Descrição. Anote o nome da política, pois ele será necessário ao criar usuários (abaixo). Escolha Create policy (Criar política) na parte inferior da janela.
6. Você será levado de volta para a janela do console do IAM, onde deverá ver um banner confirmando que sua nova política foi criada.

## Criar usuários e adicionar permissões

### Chaves de acesso do usuário do IAM

As chaves de acesso do IAM consistem em um ID de chave de acesso e em uma chave de acesso secreta. Elas são usadas para assinar as solicitações programáticas que você faz à AWS. Se não tiver chaves de acesso, será possível criá-las a partir do Console de Gerenciamento da AWS. Como prática recomendada, não crie chaves de acesso do usuário raiz.

A única vez que é possível exibir ou baixar uma chave de acesso secreta é quando você cria chaves de acesso. Não será possível recuperá-las posteriormente. Contudo, é possível criar novas chaves de acesso a qualquer momento, caso tenha as permissões para realizar as ações do IAM necessárias.

Sempre armazene as chaves de acesso com segurança. Nunca as compartilhe com terceiros (mesmo que uma consulta pareça vir da Amazon). Para obter mais informações, consulte Gerenciamento de chaves de acesso de usuários do IAM no Guia do usuário do IAM.

### Procedimento

Siga estas etapas:

1. No painel de navegação, selecione Usuários e depois Criar usuário. Uma janela Especificar detalhes do usuário é aberta.
2. Na janela Especificar detalhes do usuário:
   a. Em Detalhes do usuário, digite o novo Nome de usuário a ser criado.
   b. Marque Fornecer ao usuário acesso ao Console de Gerenciamento da AWS).
   c. Em Senha do console, selecione Senha gerada automaticamente.
   d. Marque Usuários devem criar uma nova senha no próximo login.
   e. Escolha Próximo. Uma janela Definir permissões é aberta.
3. Em Definir permissões, selecione Anexar políticas diretamente. Uma janela Políticas de permissões é aberta.
4. Na caixa de pesquisa, insira o nome de uma política do IVS (uma política gerenciada pela AWS ou sua política personalizada criada anteriormente). Quando ela for localizada, marque a caixa para selecionar a política.
5. Escolha Avançar (na parte inferior da janela). Uma janela Revisar e criar é aberta.
6. Na janela Revisar e criar, confirme que todos os detalhes do usuário estão corretos e escolha Criar usuário (na parte inferior da janela).
7. A janela Recuperar senha é aberta, contendo seus Detalhes de login no console. Salve essas informações em segurança para referência futura. Quando terminar, escolha Voltar à lista de usuários.

## Adicionar permissões para um usuário existente

Siga estas etapas:

1. Faça login no Console de Gerenciamento da AWS e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. No painel de navegação, escolha Users (Usuários) e escolha um nome de usuário existente a ser atualizado. (Escolha o nome clicando nele; não marque a caixa de seleção.)
3. Na página Resumo, na guia Permissões, escolha Adicionar permissões. Uma janela Adicionar permissões é aberta.

4. Selecione Attach existing policies directly (Anexar políticas existentes diretamente). Uma janela Políticas de permissões é aberta.
5. Na caixa de pesquisa, insira o nome de uma política do IVS (uma política gerenciada pela AWS ou sua política personalizada criada anteriormente). Quando a política for localizada, marque a caixa para selecionar a política.
6. Escolha Avançar (na parte inferior da janela). Uma janela Revisar é aberta.
7. Na janela Revisar, selecione Adicionar permissões (na parte inferior da janela).
8. Na página Summary (Resumo), confirme se a política do IVS foi adicionada.

# Criar um palco

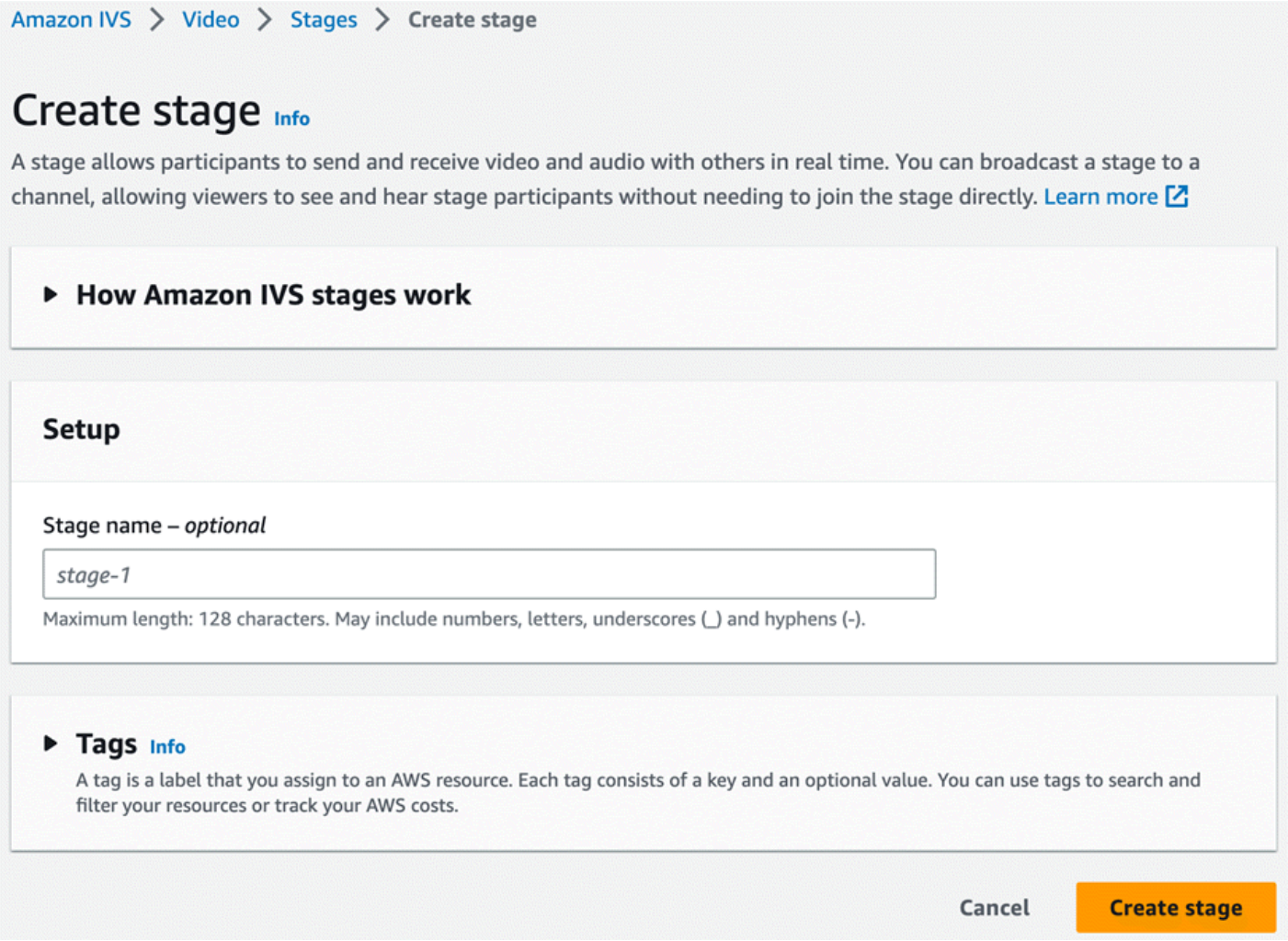
Um palco corresponde a um espaço virtual no qual os participantes podem trocar vídeos em tempo real. Ele é o recurso fundamental da API de streaming em tempo real. Você pode criar um estágio usando o console ou o CreateStage endpoint.

Recomendamos que, sempre que possível, você crie um novo palco para cada sessão lógica e exclua-o quando terminar, em vez de manter os palcos antigos para possível reutilização. Se os recursos obsoletos (palcos antigos, que não devem ser reutilizados) não forem limpos, é provável que você atinja o limite do número máximo de palcos mais rapidamente.

## Instruções do console

1. Abra o console do Amazon IVS.

   (Você também pode acessar o console do Amazon IVS via Console de Gerenciamento da AWS .)
2. No painel de navegação esquerdo, selecione Palcos e, em seguida, selecione Criar palco. A janela Criar palco é exibida.




3. Opcionalmente, insira um Nome de palco. Selecione Criar palco para criar o palco. A página de detalhes do palco é exibida para o novo palco.

## Instruções da CLI

Para instalar a AWS CLI, consulte Instalar ou atualizar para a versão mais recente da AWS CLI.

Agora é possível usar a CLI para criar e gerenciar recursos. A API do palco está sob o namespace ivs-realtime. Por exemplo, para criar um palco:

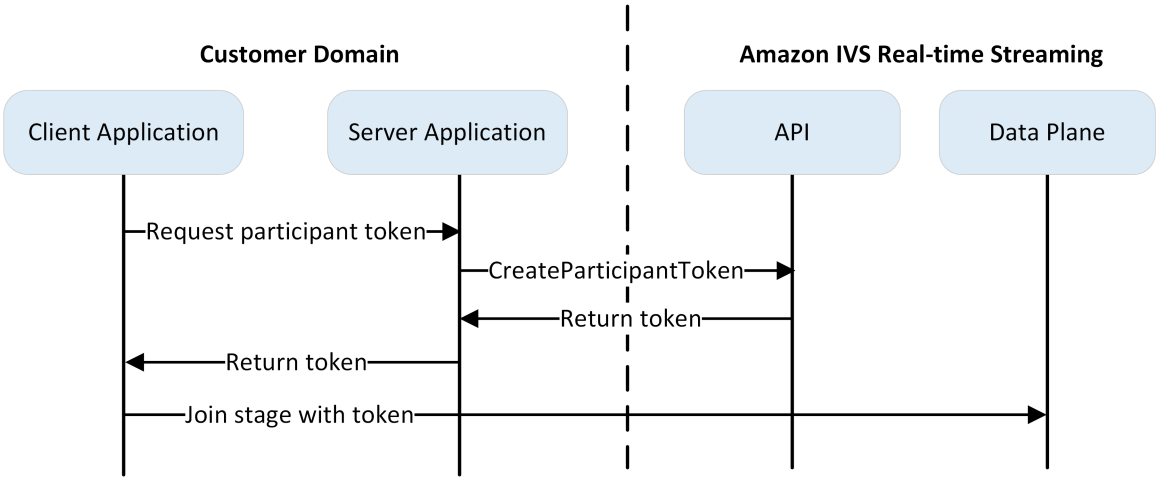
```
aws ivs-realtime create-stage --name "test-stage"
```

A resposta é:

```
{
```

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

# Distribuir tokens de participantes




Agora que você tem um palco, é necessário criar e distribuir tokens aos participantes para possibilitar que eles ingressem no palco e comecem a enviar e receber vídeos.

Conforme mostrado acima, um aplicativo cliente solicita um token ao aplicativo do servidor, e o aplicativo do servidor chama CreateParticipantToken usando uma solicitação assinada pelo AWS SDK ou SigV4. Como as credenciais da AWS são usadas para chamar a API, o token deve ser gerado em uma aplicação segura do lado do servidor, não na aplicação do lado do cliente.

Ao criar um token de participante, opcionalmente, é possível especificar as funcionalidades habilitadas por esse token. O padrão é `PUBLISH` e `SUBSCRIBE`, que permite ao participante enviar e receber áudio e vídeo, mas você pode emitir tokens com um subconjunto de funcionalidades. Por exemplo, é possível emitir um token somente com a funcionalidade `SUBSCRIBE` para moderadores. Nesse caso, os moderadores podem visualizar os participantes que estão enviando vídeos, mas não podem enviar seus próprios vídeos.

É possível criar tokens de participantes usando o console ou a CLI para testes e desenvolvimento, mas provavelmente você desejará criá-los com o AWS SDK em seu ambiente de produção.

Você precisará de uma maneira de distribuir os tokens do seu servidor para cada cliente (por exemplo, por meio de uma solicitação de API). Não fornecemos essa funcionalidade. Para este guia, você pode simplesmente copiar e colar os tokens no código do cliente nas etapas a seguir.

Importante: trate os tokens como opacos; ou seja, não desenvolva funcionalidades com base no conteúdo do token. O formato dos tokens poderá mudar no futuro.

## Instruções do console

1. Navegue até o palco criado na etapa anterior.
2. Selecione Criar um token de participante. A janela Criar um token de participante é exibida.
3. Insira um ID de usuário a ser associado ao token. Isso pode ser qualquer texto codificado em UTF-8.
4. Selecione Criar um token de participante.
5. Copie o token. Importante: certifique-se de salvar o token. O IVS não o armazena, e você não poderá recuperá-lo posteriormente.

## Instruções da CLI

A criação de um token com a AWS CLI requer que você faça o download e configure a CLI em sua máquina primeiro. Para obter mais detalhes, consulte o Guia do usuário da Interface de Linhas de Comando da AWS. Observe que gerar tokens com a AWS CLI é bom para fins de testes, mas para uso em produção, recomendamos que você gere tokens do lado do servidor com o AWS SDK (consulte as instruções abaixo).

1. Execute o comando `create-participant-token` com o ARN do palco. Inclua qualquer uma ou todas as seguintes funcionalidades: `"PUBLISH"` e `"SUBSCRIBE"`.

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

2. Isso retorna um token de participante:

```
{
    "participantToken": {
        "capabilities": [
            "PUBLISH",
            "SUBSCRIBE"
        ],
        "expirationTime": "2023-06-03T07:04:31+00:00",
        "participantId": "tU06DT5jCJeb",
```

```
        "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtdmVFTm9sUyIsInJ
TaKjllW9Qac6c5xBrdAk"    }
}
```

3. Salve esse token. Você precisará dele para ingressar no palco e enviar e receber vídeos.

## Instruções do AWS SDK

É possível usar o AWS SDK para criar tokens. Abaixo estão as instruções de uso JavaScript do AWS SDK.

Importante: esse código deve ser executado no lado do servidor e sua saída transferida para o cliente.

Pré-requisito: Para usar o exemplo de código abaixo, você precisa instalar o pacote client-ivs-realtime aws-sdk/. Para obter detalhes, consulte Introdução ao SDK da AWS 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);
```

# Integrar o SDK de Transmissão do IVS

O IVS fornece um SDK de transmissão para Web, Android e iOS que você pode integrar à sua aplicação. O SDK de transmissão é usado para enviar e receber vídeos. Nesta seção, escrevemos uma aplicação simples que possibilita que dois ou mais participantes interajam em tempo real. As etapas abaixo orientam você na criação de um aplicativo chamado BasicRealTime. O código completo do aplicativo está ativado CodePen e 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

## Configurar arquivos

Para começar a configurar seus arquivos, crie uma pasta e um arquivo HTML e JS inicial:

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

É possível instalar o SDK de transmissão usando uma etiqueta de script ou um npm. Nosso exemplo usa a etiqueta de script para simplificar, mas é fácil de modificar se você optar por usar o npm posteriormente.

## Uso de uma etiqueta de script

O SDK de transmissão da Web é distribuído como uma JavaScript biblioteca e pode ser recuperado em https://web-broadcast.live-video.net/1.8.0/ amazon-ivs-web-broadcast .js.

Quando carregada via etiqueta `<script>`, a biblioteca expõe uma variável global no escopo da janela chamada `IVSBroadcastClient`.

## Uso de npm

Para instalar o pacote npm:

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

Agora você pode acessar o BroadcastClient objeto IVS:

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

# Android

## Criar o projeto Android

1. No Android Studio, crie um New Project.
2. Escolha Empty Views Activity.

Observação: em algumas versões mais antigas do Android Studio, a atividade baseada em visualizações é chamada de Empty Activity. Se a janela do Android Studio mostrar Empty Activity e não mostrar Empty Views Activity, selecione Empty Activity. Caso contrário, não selecione Empty Activity, pois usaremos APIs de visualização (não o Jetpack Compose).

3. Em Name, coloque o nome do seu projeto e, em seguida, selecione Finish.

### Instalar o SDK de transmissão

Para adicionar a biblioteca de transmissão do Amazon IVS para Android ao seu ambiente de desenvolvimento do Android, adicione a biblioteca ao arquivo `build.gradle` do módulo, conforme mostrado aqui (para a versão mais recente do SDK de transmissão do Amazon IVS). Em projetos mais recentes, o repositório `mavenCentral` pode já estar incluso em seu arquivo `settings.gradle`. Se for esse o caso, você pode omitir o bloco `repositories`. Para nossa amostra, também precisaremos habilitar a associação de dados no bloco `android`.

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
}

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

Como alternativa, para instalar o SDK manualmente, baixe a versão mais recente neste local:

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

## iOS

### Criar o projeto iOS

1. Crie um novo projeto no Xcode.
2. Em Platform, selecione iOS.
3. Em Application, selecione App.

4. Em Product Name, insira o nome de produto da sua aplicação e, em seguida, selecione Next.
5. Escolha (navegue até) um diretório no qual deseja salvar o projeto e, em seguida, selecione Create.

Em seguida, é necessário trazer o SDK. Recomendamos que você integre o SDK de transmissão via CocoaPods. Como alternativa, é possível adicionar manualmente a estrutura ao seu projeto. Ambos os métodos são descritos abaixo.

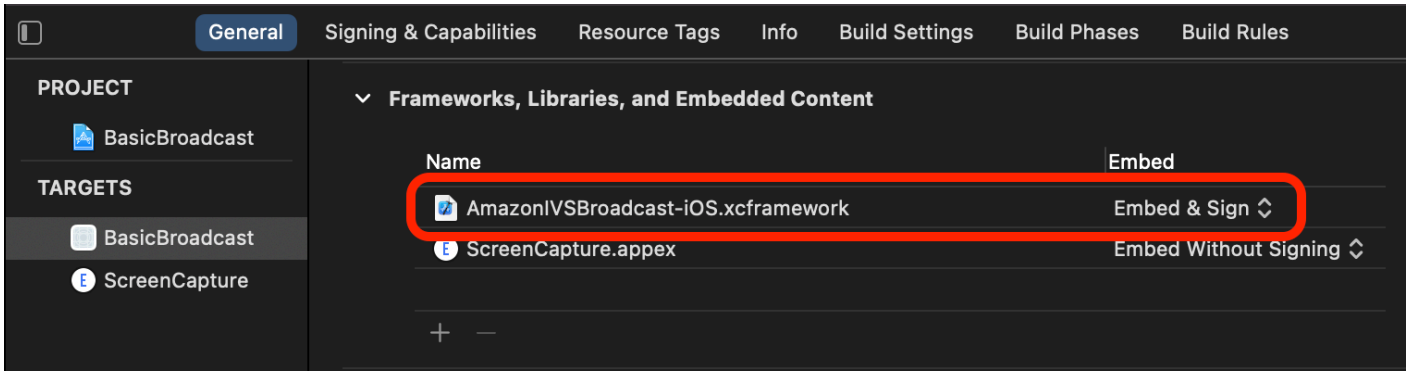
### Recomendado: instale o Broadcast SDK () CocoaPods

Supondo que o nome do seu projeto seja `BasicRealTime`, crie um `Podfile` na pasta do projeto com o seguinte conteúdo e, em seguida, execute `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
```

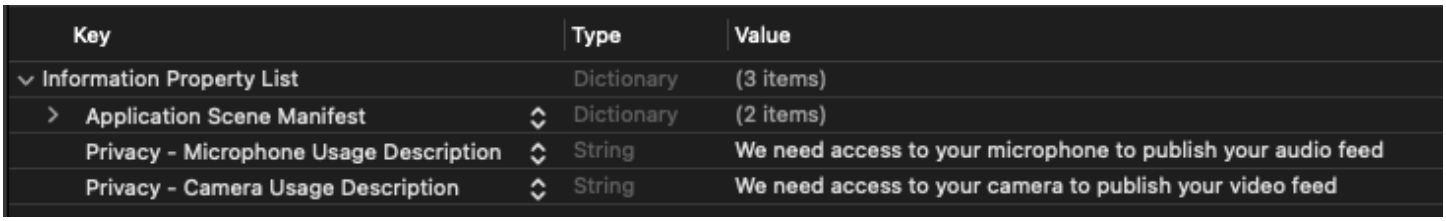
### Abordagem alternativa: instalar o framework manualmente

1. Baixe a versão mais recente em https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Extraia o conteúdo do arquivo. `AmazonIVSBroadcast.xcframework` contém o SDK para dispositivo e para o simulador.
3. Incorporar o `AmazonIVSBroadcast.xcframework` arrastando-o para a seção Estruturas, bibliotecas e conteúdo incorporado da guia Geral para o destino da sua aplicação:



### Configurar permissões

Você precisa atualizar o `Info.plist` do seu projeto para adicionar duas novas entradas para `NSCameraUsageDescription` e `NSMicrophoneUsageDescription`. Para os valores, inclua explicações para o usuário sobre por que sua aplicação está solicitando acesso à câmera e ao microfone.




## Publicar e inscrever-se em vídeos

Veja os detalhes abaixo para web, Android e iOS.

## Web

### Criar um padrão em HTML

Primeiro, criaremos o padrão em HTML e importaremos a biblioteca como uma etiqueta de 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>
```

### Aceitar entrada de tokens e adicionar botões Ingressar/Sair

Aqui, preencheremos o corpo com nossos controles de entrada. Eles recebem o token como entrada e configuram os botões Ingressar e Sair. Normalmente, as aplicações solicitarão o token da API da sua aplicação, mas, para este exemplo, você copiará e colará o token na entrada do token.

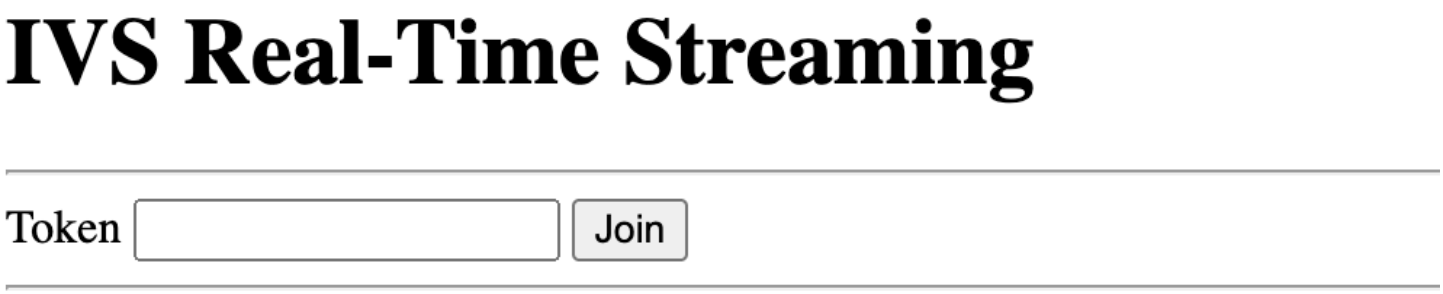
```
<h1>IVS Real-Time Streaming</h1>
<hr />

<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 />
```

### Adicionar elementos de contêiner de mídia

Esses elementos manterão a mídia para nossos participantes locais e remotos. Adicionamos uma etiqueta de script para carregar a lógica da nossa aplicação definida em `app.js`.

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

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

<!-- Load Script -->
<script src="./app.js"></script>
```

Isso completa a página HTML, e você verá o seguinte ao carregar `index.html` em um navegador:



## Criar app.js

Agora definiremos o conteúdo do nosso arquivo `app.js`. Comece com a importação de todas as propriedades necessárias do SDK global:

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

## Criar variáveis da aplicação

Estabeleça variáveis para manter referências aos nossos elementos HTML dos botões Ingressar e Sair e armazenar o estado para a aplicação:

```
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;
```

## Criar joinStage 1: definir a função e validar a entrada

A função `joinStage` recebe o token de entrada, cria uma conexão com o palco e começa a publicar o vídeo e o áudio recuperados de `getUserMedia`.

Para começar, definimos a função e validamos o estado e a entrada do token. Desenvolveremos essa função nas próximas seções.

```
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
};
```

## Criar joinStage 2: obter mídia para publicação

Aqui está a mídia que será publicada no palco:

```
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]);
```

## Criar joinStage 3: definir a estratégia do palco e criar o palco

Essa estratégia de palco corresponde ao ponto central da lógica de decisão que o SDK usa para definir o que publicar e em quais participantes se inscrever. Para obter mais informações sobre a finalidade da função, consulte Strategy.

Essa estratégia é simples. Após ingressar no palco, publique os streams que acabamos de recuperar e inscreva-se nos áudios e vídeos de todos os participantes remotos:

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

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

## Criar joinStage 4: lidar com eventos de palco e renderizar mídia

Os palcos emitem muitos eventos. Precisamos receber `STAGE_PARTICIPANT_STREAMS_ADDED` e `STAGE_PARTICIPANT_LEFT` para renderizar mídia e removê-la da página. Um conjunto mais completo de eventos está listado em Eventos.

Observe que criamos quatro funções auxiliares aqui para nos ajudar no gerenciamento dos elementos DOM necessários: `setupParticipant`, `teardownParticipant`, `createVideoEl` e `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;
}
```

## Criar joinStage 5: ingressar no palco

Concluiremos nossa função `joinStage` ao finalmente ingressar no palco.

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

## Criar leaveStage

Defina a função `leaveStage` que o botão Sair invocará.

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

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

## Inicializar manipuladores de eventos de entrada

Adicionaremos uma última função ao nosso arquivo `app.js`. Essa função é invocada imediatamente quando a página é carregada e estabelece manipuladores de eventos para ingressar e sair do palco.

```
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
```

### Executar a aplicação e fornecer um token

Nesse ponto, você pode compartilhar a página da Web localmente ou com outras pessoas, abrir a página, inserir um token de participante e ingressar no Stage.

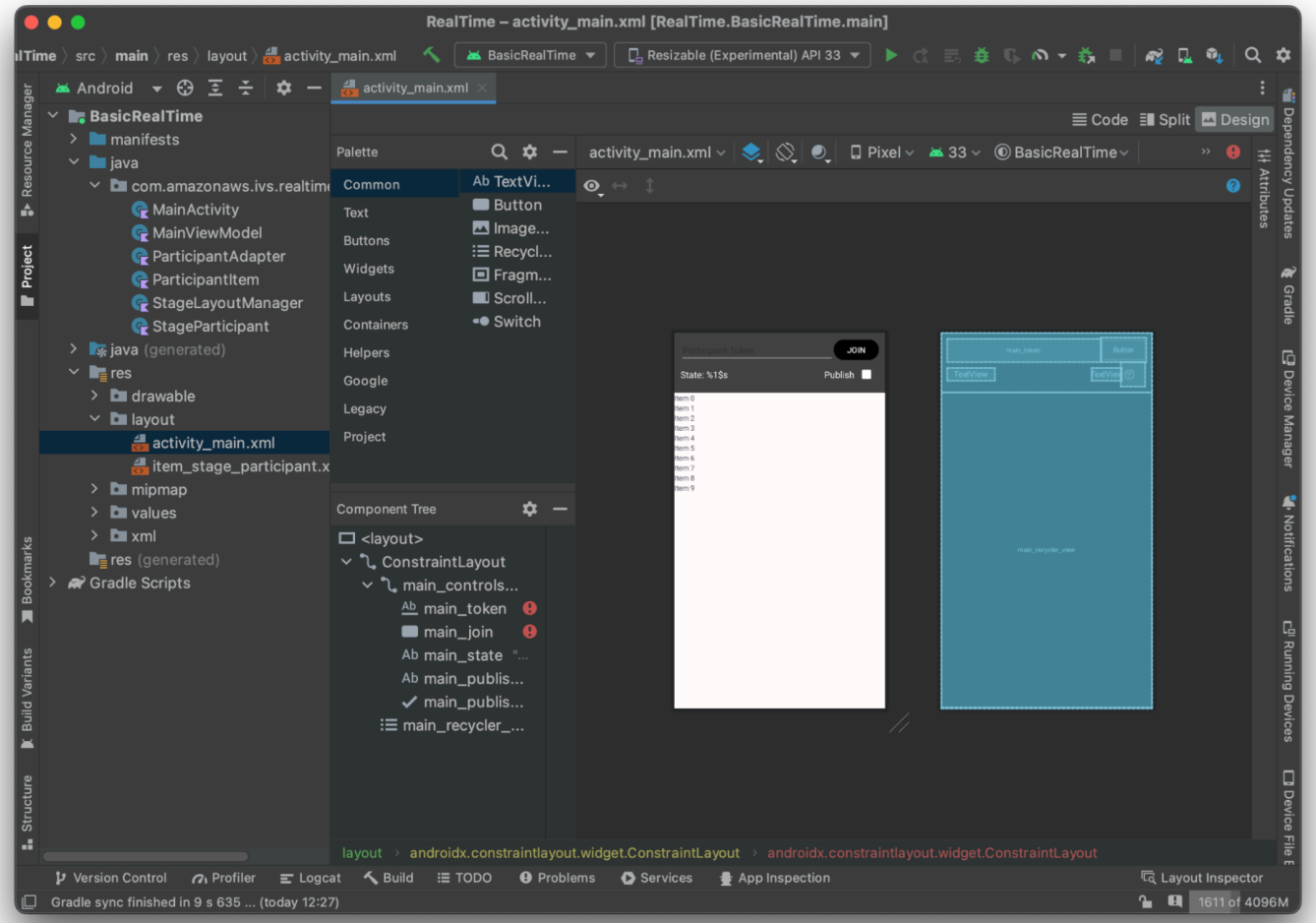
### E depois?

Para obter exemplos mais detalhados envolvendo npm, React etc, consulte SDK de Transmissão do IVS: Guia do Android (streaming em tempo real).

## Android

### Criar visualizações

Começamos com a criação de um layout simples para nossa aplicação usando o arquivo `activity_main.xml` criado automaticamente. O layout contém um `EditText` para adicionar um token, um `Button` Ingressar, um `TextView` para mostrar o estado do palco e um `CheckBox` para alternar a publicação.




Este é o XML por trás da visualização:

```
<?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>
```

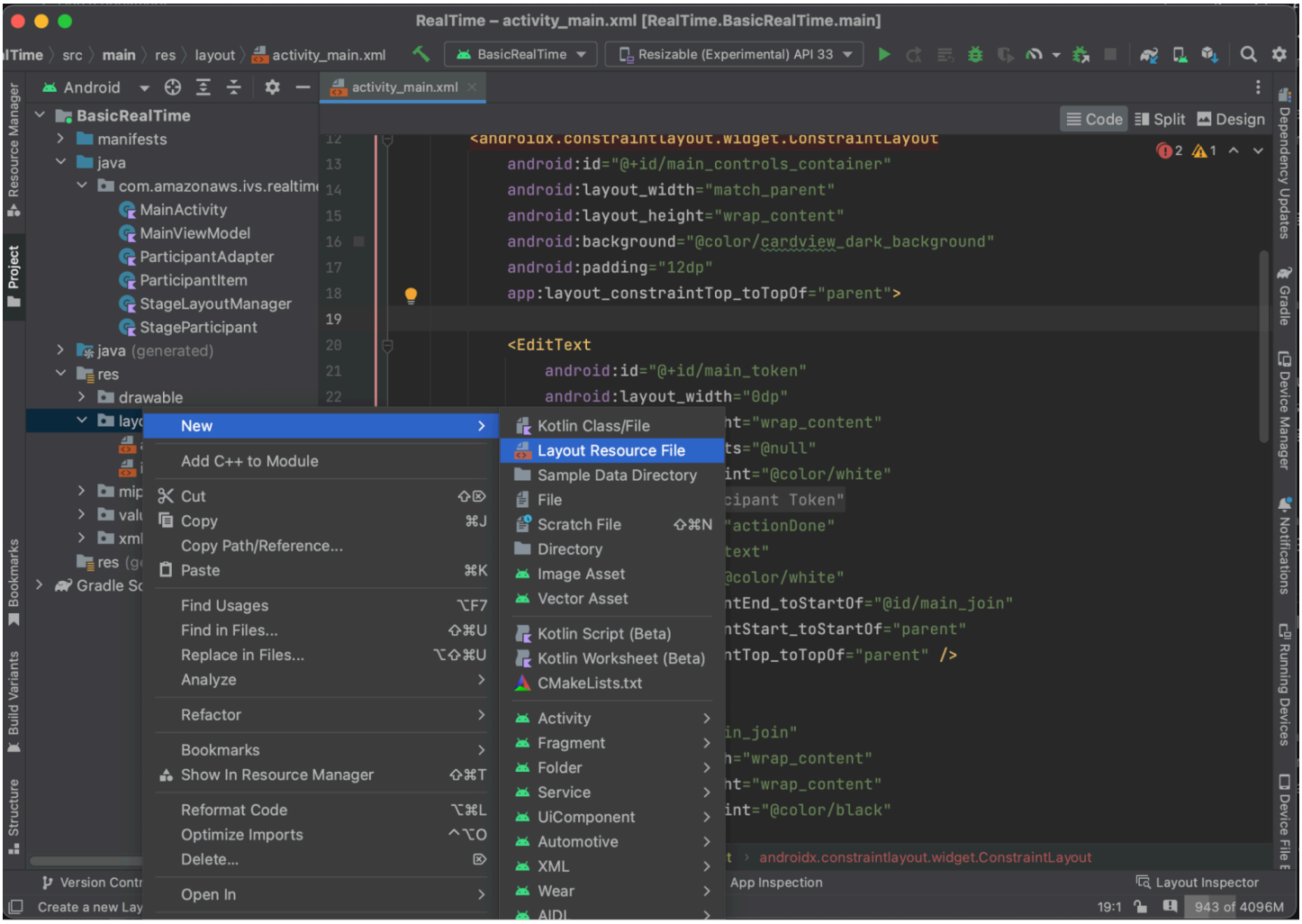
Nós referenciamos alguns IDs de string aqui, então criaremos nosso arquivo `strings.xml` completo agora:

```
<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>
```

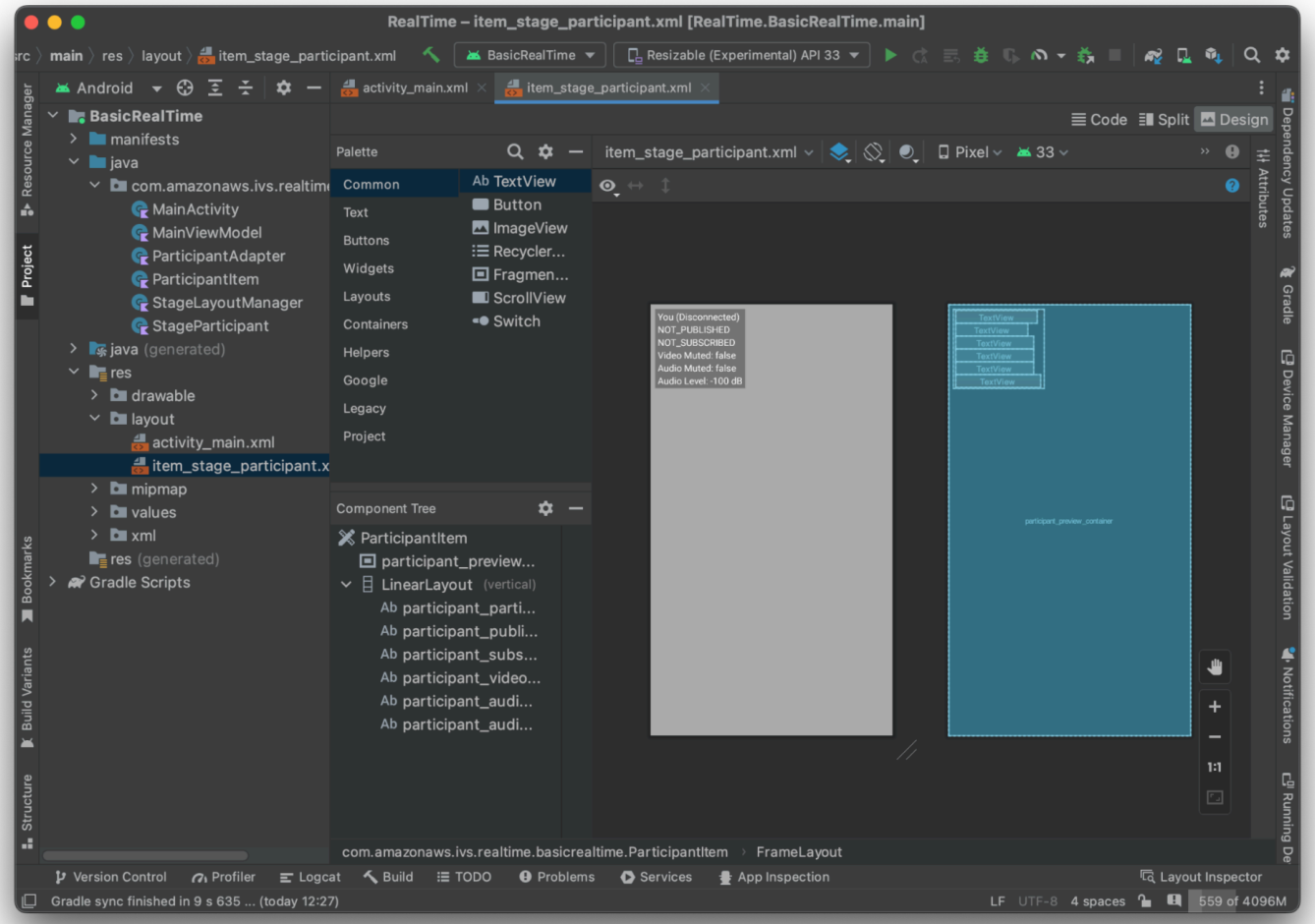
Vincularemos essas visualizações no XML à nossa `MainActivity.kt`:

```
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)
}
```

Agora, criamos uma visualização do item para nosso `RecyclerView`. Para fazer isso, clique com o botão direito do mouse em seu diretório `res/layout` e selecione New > Layout Resource File. Nomeie esse novo arquivo como `item_stage_participant.xml`.




O layout desse item é simples. Ele contém uma visualização para renderizar o stream de vídeo de um participante e uma lista de rótulos para exibir informações sobre o participante:




Este é o 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 arquivo XML infla uma classe que ainda não criamos, `ParticipantItem`. Como o XML inclui o namespace completo, certifique-se de atualizar esse arquivo XML com o seu namespace. Criaremos esta classe e configuraremos as visualizações posteriormente, por isso deixe em branco por enquanto.

Crie uma nova classe 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)
    }
}
```

## Permissões

Para usar a câmera e o microfone, você precisa solicitar permissões por parte do usuário. Para isso, seguiremos um fluxo de permissões padrão:

```
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 da aplicação

Nosso aplicativo acompanha os participantes localmente em um `MainViewModel.kt` e o estado será comunicado de volta ao `MainActivity` usando o Kotlin. StateFlow

Crie uma nova classe 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 {

}
```

Em `MainActivity.kt`, gerenciamos nosso modelo de visualização:

```
import androidx.activity.viewModels

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

Para usar `AndroidViewModel` e essas extensões `ViewModel` do Kotlin, será necessário adicionar o seguinte ao arquivo `build.gradle` do seu 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

Criaremos uma subclasse `RecyclerView.Adapter` simples para acompanhar nossos participantes e atualizar nosso `RecyclerView` em relação aos eventos do palco. Porém, antes precisamos de uma classe que represente um participante. Crie uma nova classe 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 essa classe na classe `ParticipantAdapter` que criaremos a seguir. Começamos com a definição da classe e com a criação de uma variável para acompanhar os 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>()
```

Também temos que definir nosso `RecyclerView.ViewHolder` antes de implementar o restante das substituições:

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

Usando isso, podemos implementar as substituições `RecyclerView.Adapter` padrão:

```
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 fim, adicionamos novos métodos que chamaremos de nosso `MainViewModel` quando forem feitas alterações nos participantes. Esses métodos correspondem a operações CRUD padrão no 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])
    }
}
```

De volta ao `MainViewModel`, é necessário criar e manter uma referência a este adaptador:

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

## Estado do estágio

Também precisamos acompanhar algum estado do palco no `MainViewModel`. Definiremos essas propriedades agora:

```
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 sua própria visualização prévia antes de ingressar em um palco, criamos um participante local imediatamente:

```
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)
}
```

Precisamos ter certeza de limpar esses recursos quando o nosso `ViewModel` for limpo. Substituímos on`Cleared()` imediatamente, para não esquecermos de limpar esses recursos.

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

Agora preenchemos nossa propriedade `streams` local assim que as permissões forem concedidas, implementando o método `permissionsGranted` que chamamos 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)
    }
}
```

## Implementação do SDK do palco

Existem três conceitos principais que fundamentam a funcionalidade em tempo real: palco, estratégia e renderizador. O objetivo do projeto é minimizar a quantidade de lógica do lado do cliente que é necessária para desenvolver um produto funcional.

### Stage.Strategy

Nossa implementação de `Stage.Strategy` é simples:

```
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
}
```

Para resumir, publicamos com base em nosso estado `publishEnabled` interno e, se publicarmos, usaremos os streams que coletamos anteriormente. Por fim, para esta amostra, sempre nos inscrevemos em outros participantes, recebendo seus áudios e vídeos.

StageRenderer

A implementação de `StageRenderer` também é bastante simples, embora, devido ao número de funções, contenha um pouco mais de código. A abordagem geral neste renderizador é atualizar nosso `ParticipantAdapter` quando o SDK nos notifica sobre uma alteração em um participante. Existem certos cenários nos quais lidamos com os participantes locais de maneira diferente, pois decidimos gerenciá-los nós mesmos para que eles possam ver a visualização prévia da câmera antes de ingressar.

```
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) {}
}
```

## Implementando um personalizado RecyclerView LayoutManager

A organização de diferentes números de participantes pode ser complexa. Você deseja que eles ocupem todo o quadro da visualização primária, mas não quer lidar com a configuração de cada participante independentemente. Para facilitar isso, abordaremos a implementação de um `RecyclerView.LayoutManager`.

Crie outra nova classe, `StageLayoutManager`, que deve abranger o `GridLayoutManager`. Essa classe foi projetada para calcular o layout de cada participante com base no número de participantes em um layout de linha/coluna com base no fluxo. Cada linha tem a mesma altura que as outras, mas as colunas podem ter larguras diferentes por linha. Consulte o comentário do código acima da variável `layouts` para obter uma descrição de como personalizar esse comportamento.

```
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
}
```

De volta a `MainActivity.kt`, é necessário definir o adaptador e o gerenciador de layout para o nosso `RecyclerView`:

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

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

## Conexão de ações da interface do usuário

Estamos quase finalizando, existem apenas algumas ações da interface do usuário que precisamos conectar.

Primeiro, faremos com que nossa `MainActivity` observe as alterações de `StateFlow` do `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)
        }
    }
}
```

Em seguida, adicionamos receptores ao nosso botão Ingressar e à caixa de seleção Publicar:

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

Ambas as funcionalidades de chamada acima estão em nosso `MainViewModel`, que implementamos agora:

```
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
}
```

## Renderização dos participantes

Por fim, precisamos renderizar os dados que recebemos do SDK no item do participante que criamos anteriormente. Já temos a lógica do `RecyclerView` finalizada, então só precisamos implementar a API `bind` em `ParticipantItem`.

Começaremos com a adição da função vazia e, em seguida, abordaremos ela detalhadamente:

```
fun bind(participant: StageParticipant) {

}
```

Primeiro, trataremos do estado fácil, do ID do participante, do estado de publicação e do estado de inscrição. Para esses, apenas atualizamos nosso `TextViews` diretamente:

```
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
```

Em seguida, atualizaremos os estados de áudio e vídeo silenciados. Para obter o estado silenciado, precisamos encontrar o `ImageDevice` e o `AudioDevice` na matriz de streams. Para otimizar a performance, lembramos os últimos IDs de dispositivo anexados.

```
// 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 fim, desejamos renderizar uma visualização prévia para `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
```

Além disso, exibimos estatísticas de áudio do `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
```

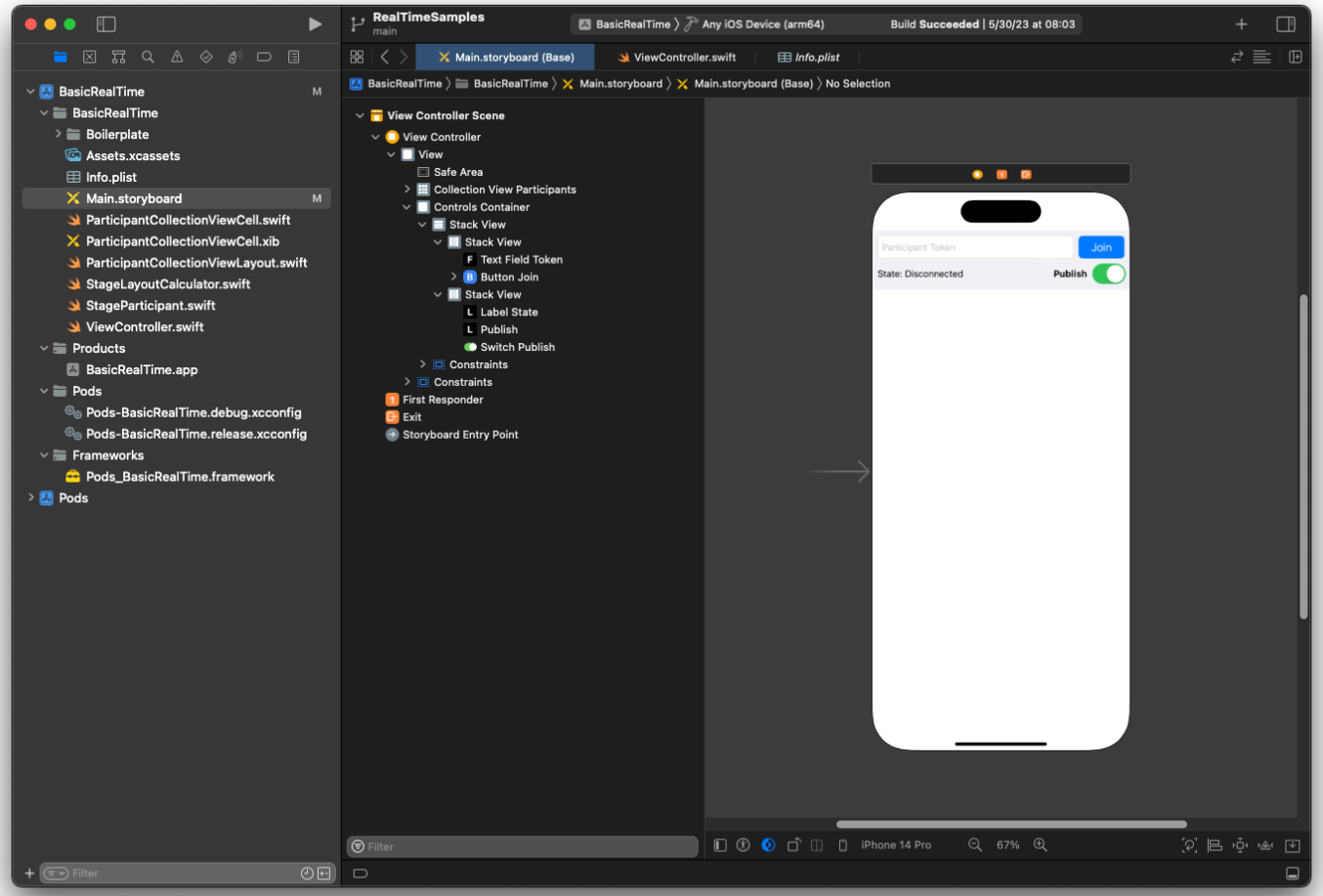
# iOS

## Criar visualizações

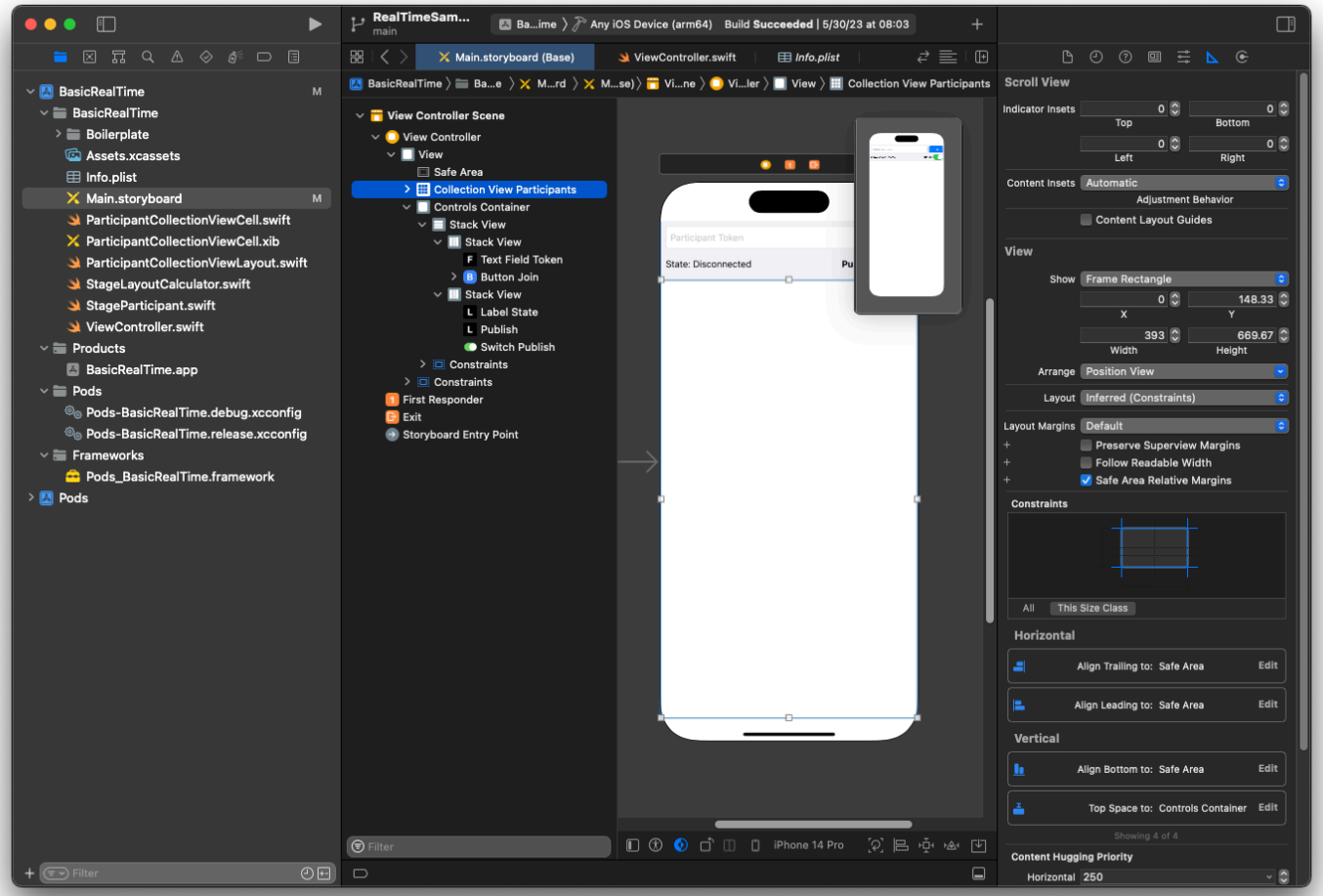
Começamos usando o arquivo `ViewController.swift` criado automaticamente para importar `AmazonIVSBroadcast` e, em seguida, adicionamos alguns `@IBOutlets` para vincular:

```
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!
```

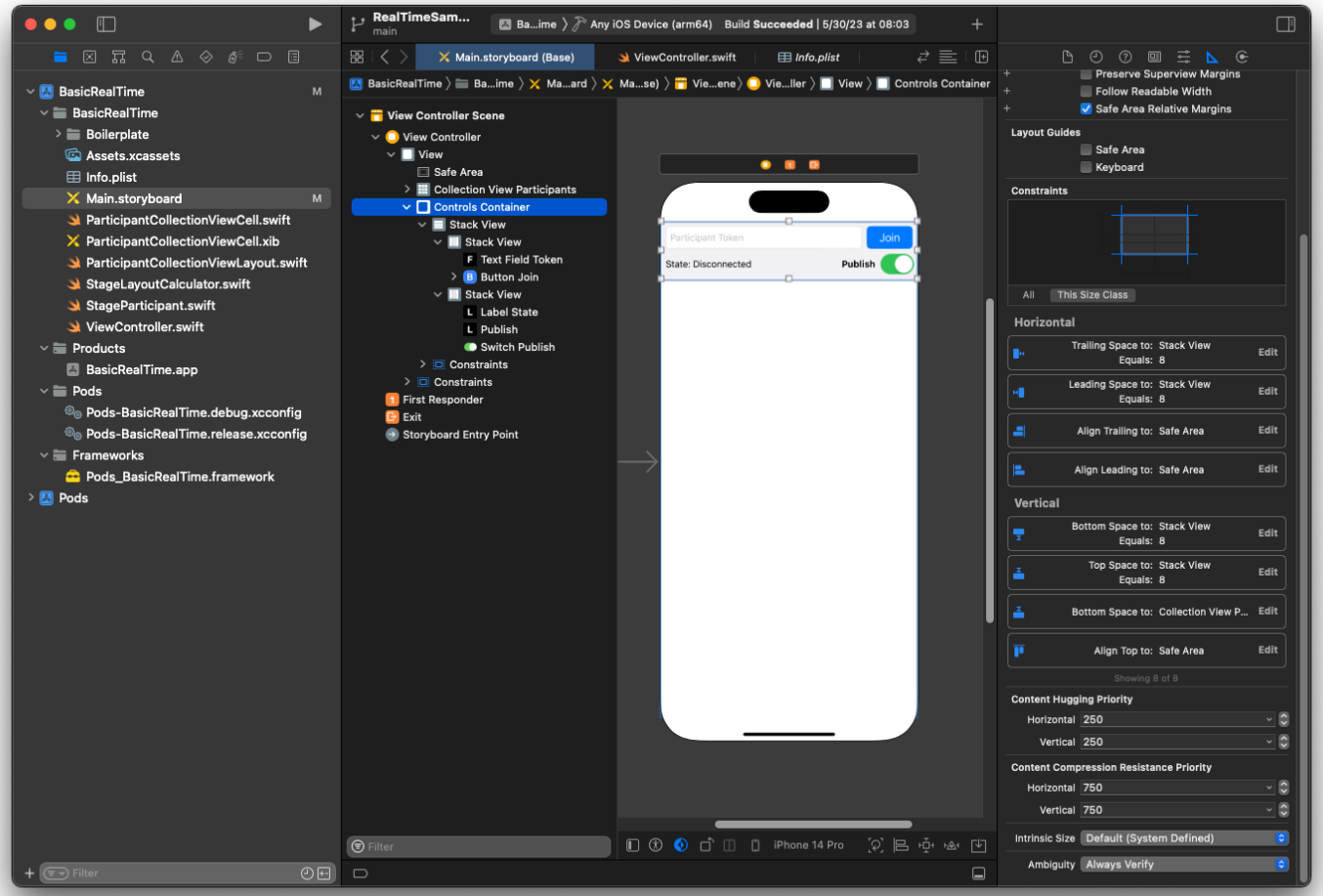
Agora, criamos essas visualizações e as vinculamos no `Main.storyboard`. Esta é a estrutura de visualização que usaremos:



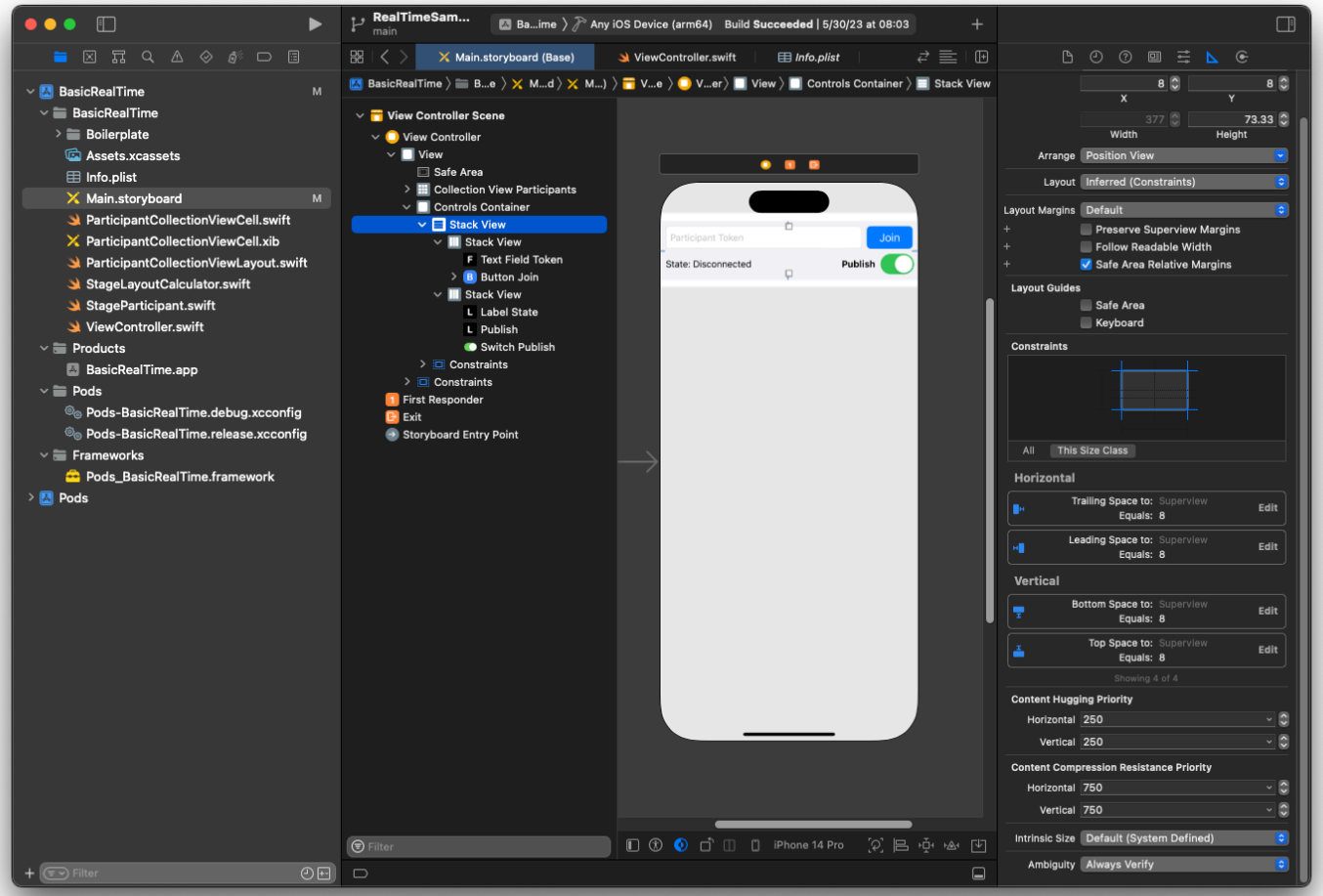

Para AutoLayout configuração, precisamos personalizar três visualizações. A primeira visualização corresponde a Collection View Participants (uma `UICollectionView`). Vincule Leading, Trailing e Bottom à Safe Area. Também vincule Top ao Controls Container.



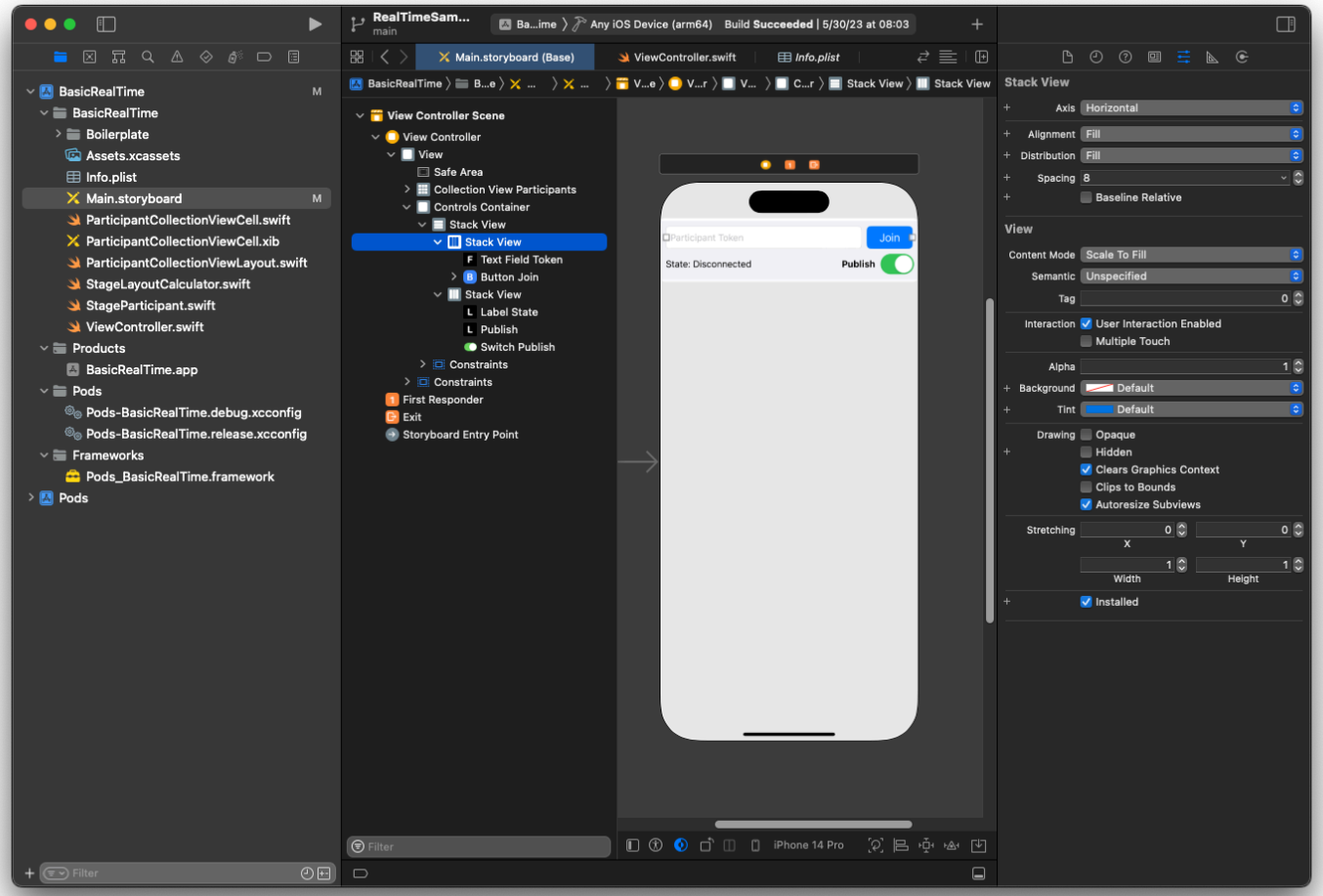

A segunda visualização corresponde ao Controls Container. Vincule Leading, Trailing e Top à Safe Area:




A terceira e última visualização corresponde a Vertical Stack View. Vincule Top, Leading, Trailing e Bottom à Superview. Para estilizar, defina o espaçamento para 8, em vez de 0.



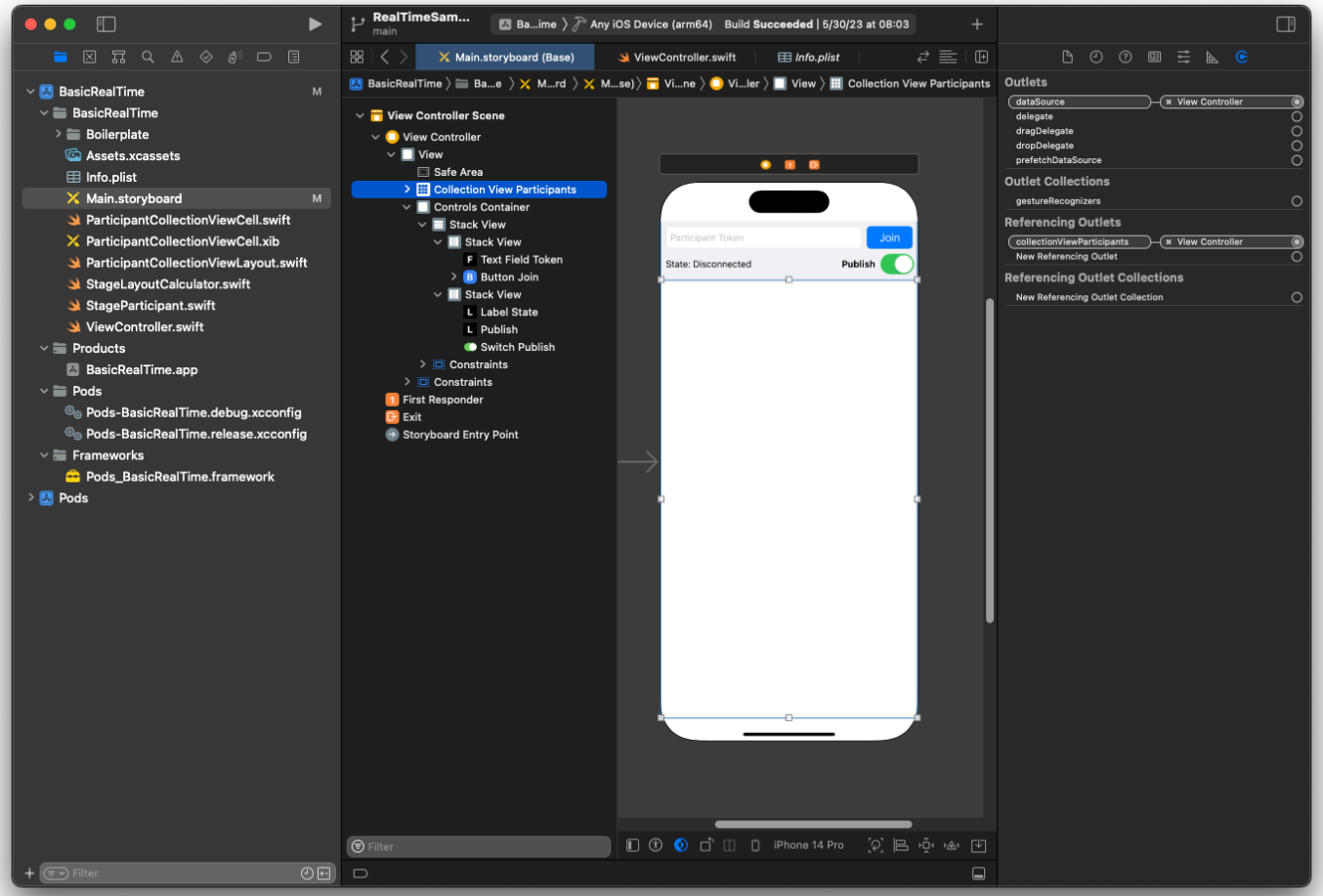

A interface do usuário StackViews cuidará do layout das visualizações restantes. Para todas as três interfaces de usuário StackViews, use Preencher como alinhamento e distribuição.



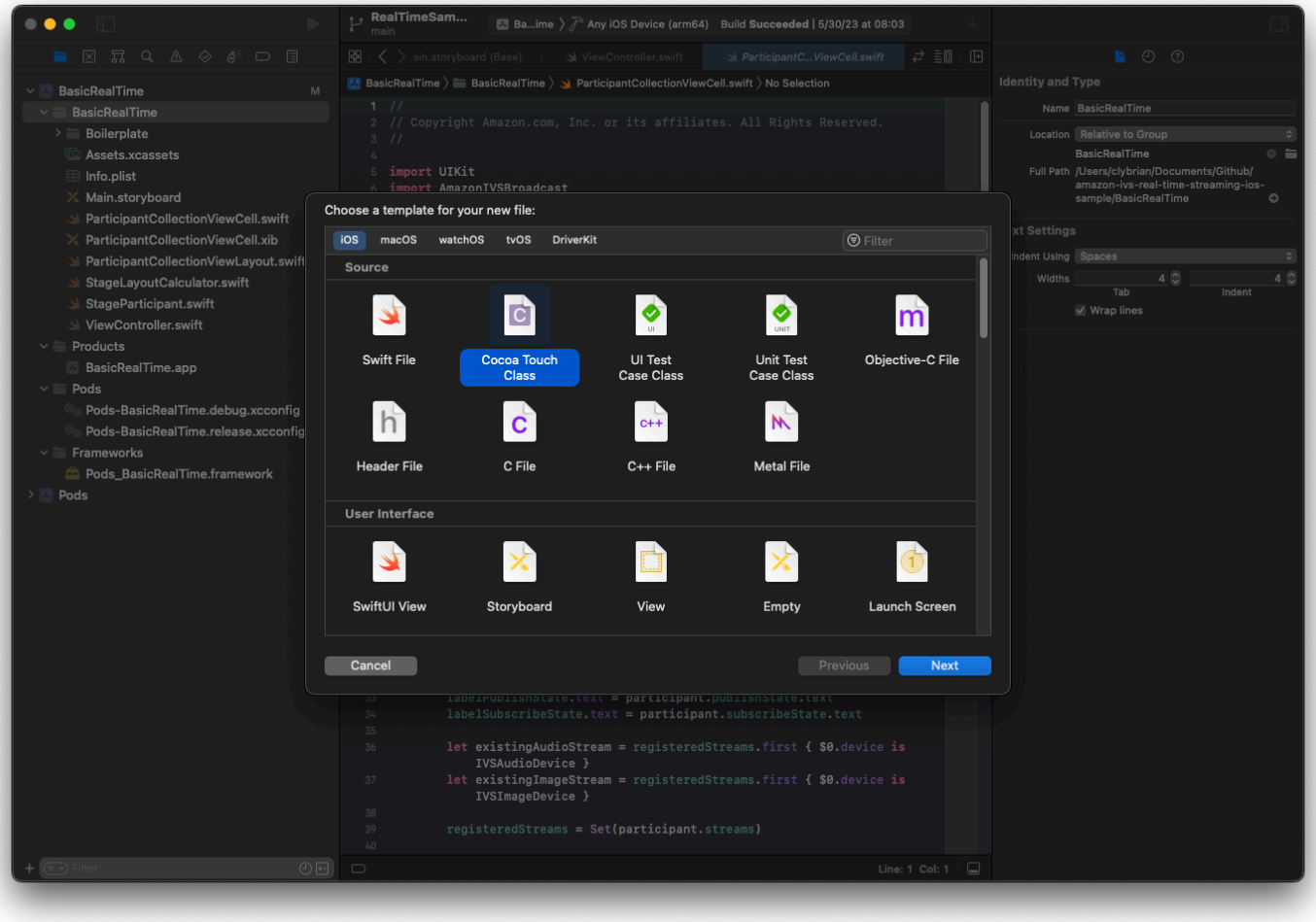

Por fim, vincularemos essas visualizações ao nosso `ViewController`. Depois da etapa acima, mapeie as seguintes visualizações:

- Text Field Join é vinculado a `textFieldToken`.
- Button Join é vinculado a `buttonJoin`.
- Label State é vinculado a `labelState`.
- Switch Publish é vinculada a `switchPublish`.
- Collection View Participants é vinculada a `collectionViewParticipants`.

Aproveite também para definir a `dataSource` do item Collection View Participants para o `ViewController` proprietário:



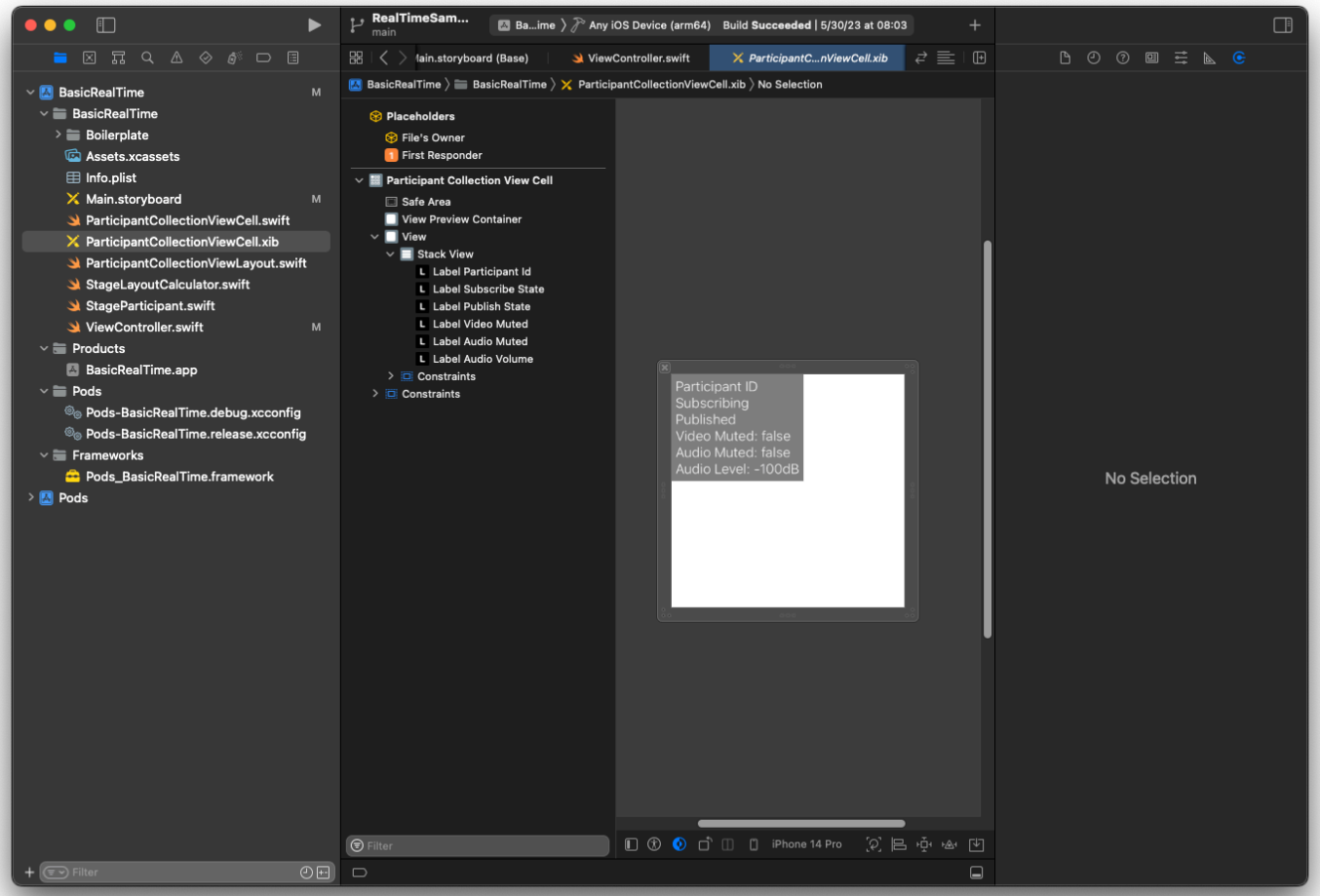

Agora, criaremos a subclasse `UICollectionViewCell` na qual renderizaremos os participantes. Comece com a criação de um novo arquivo Cocoa Touch Class:



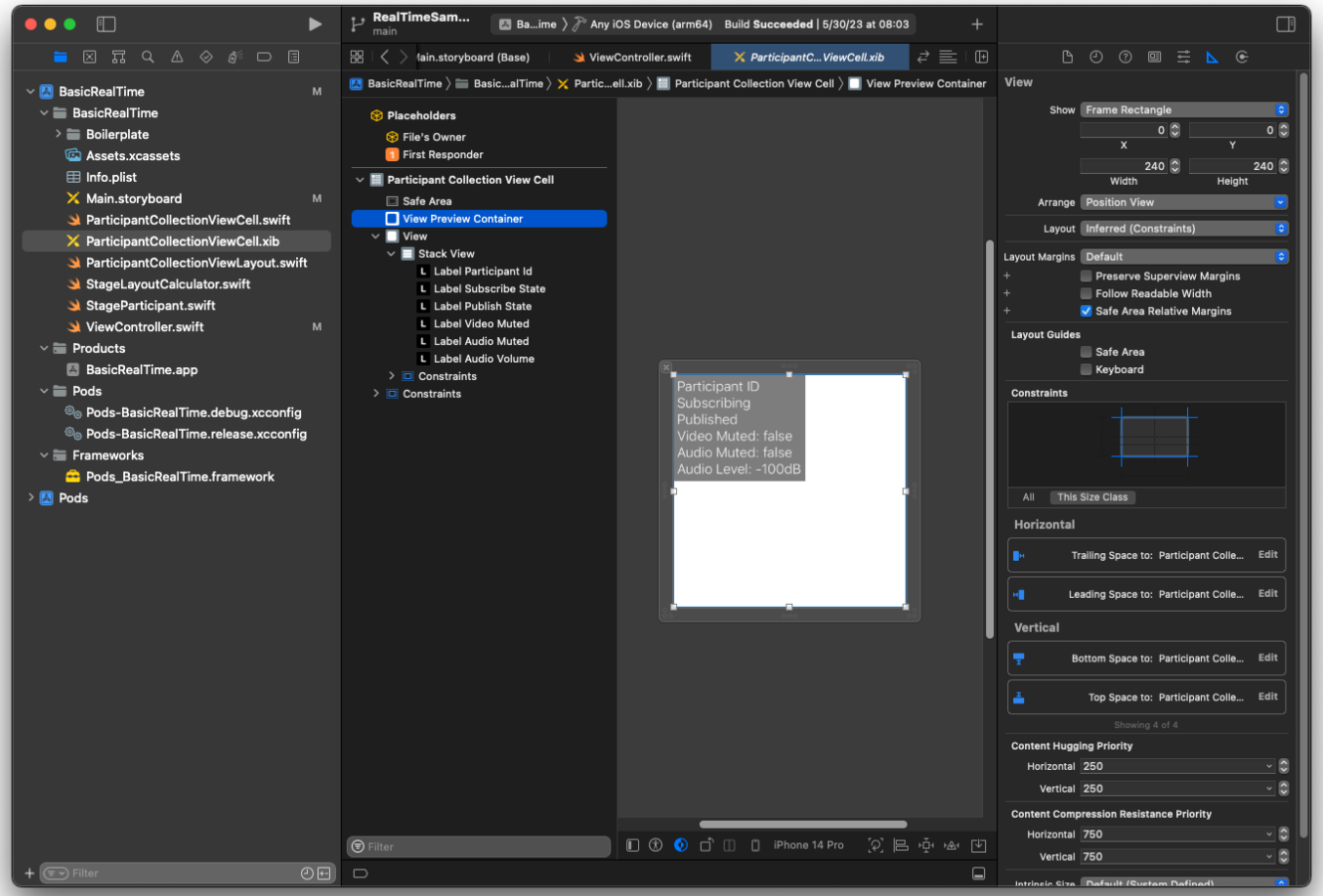

Nomeie-o como `ParticipantUICollectionViewCell` e torne-o uma subclasse de `UICollectionViewCell` no Swift. Começamos no arquivo Swift novamente, criando nosso `@IBOutlets` para vincular:

```
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!
```

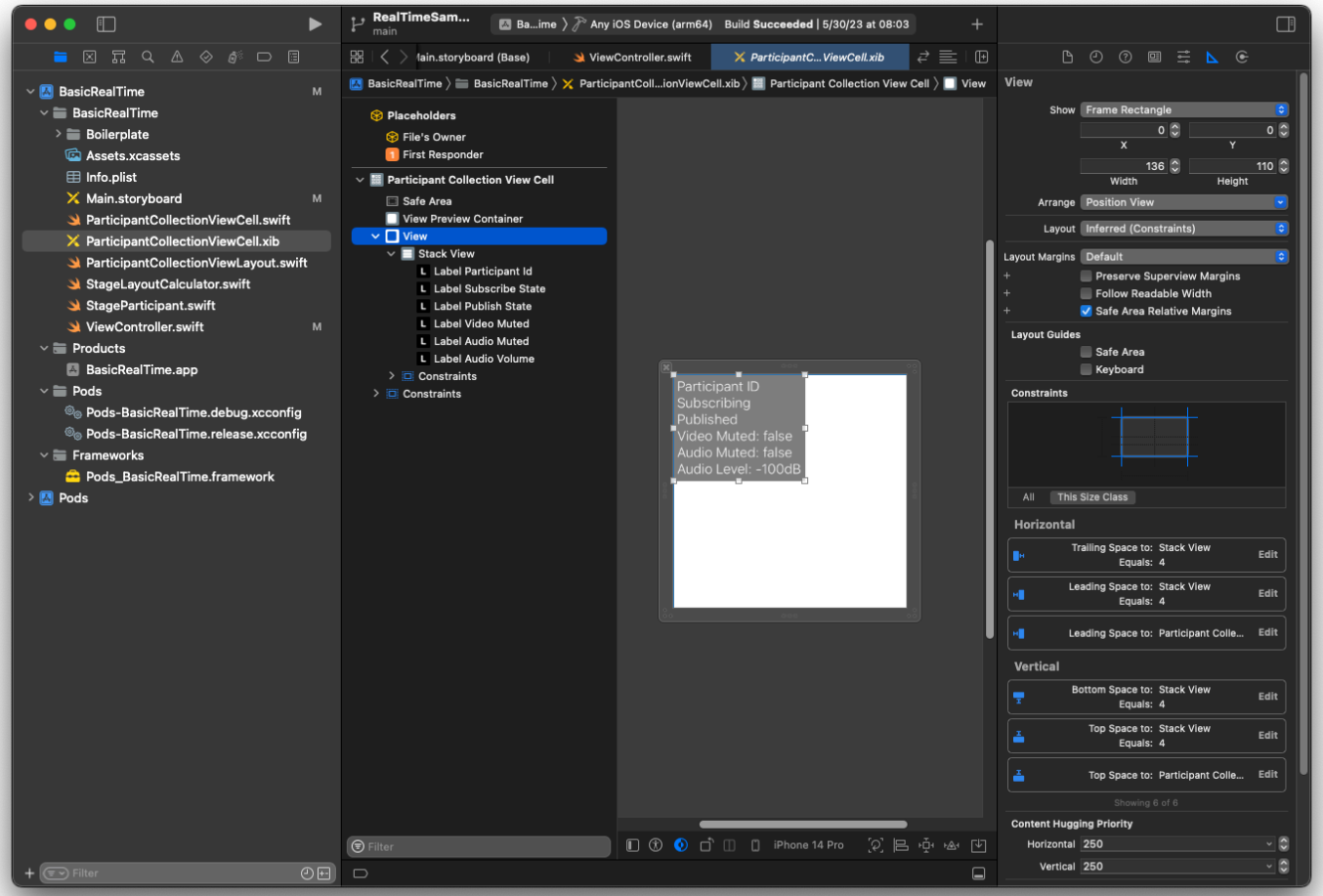
No arquivo XIB associado, crie esta hierarquia de visualização:



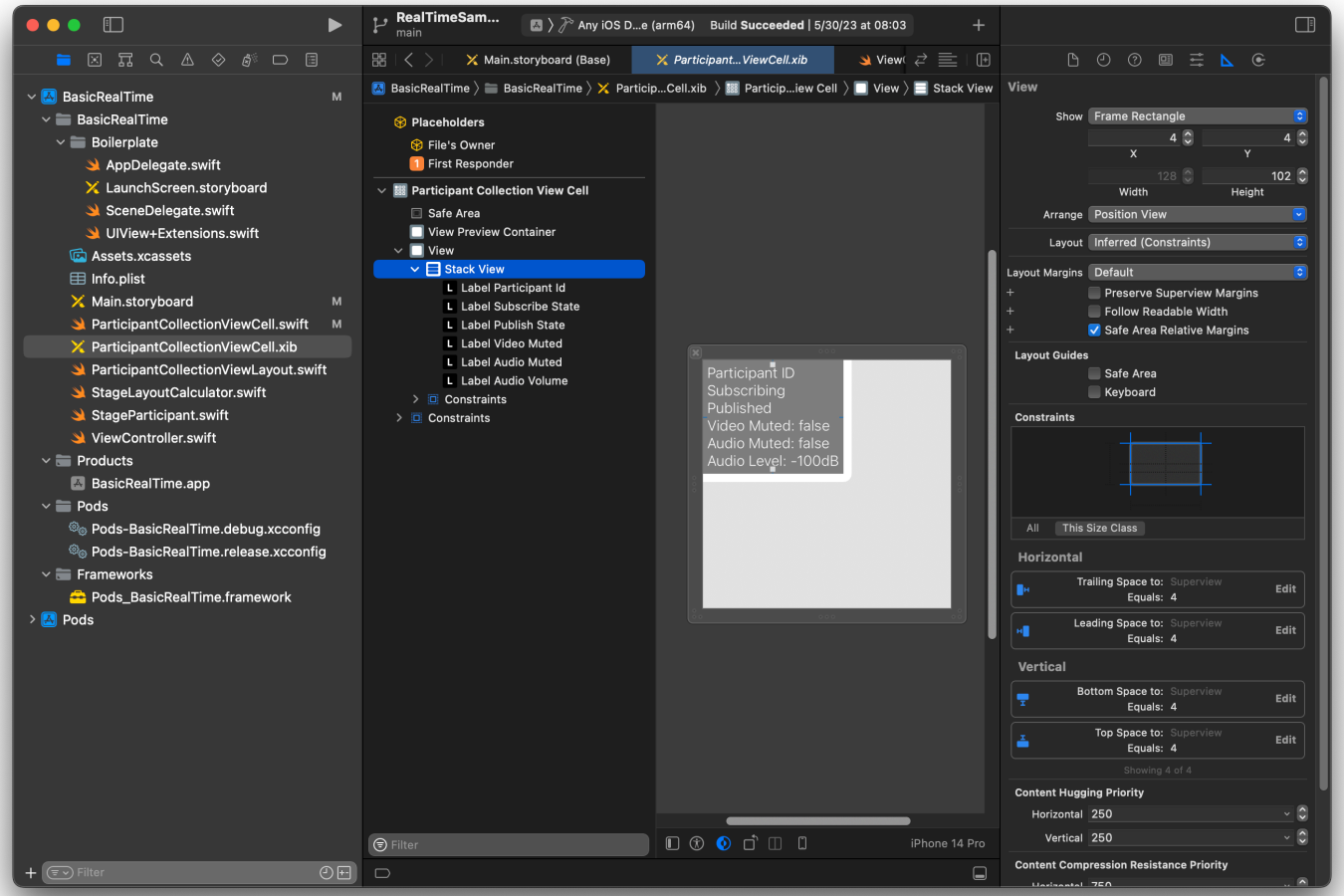

Pois AutoLayout, modificaremos três visualizações novamente. A primeira visualização corresponde a View Preview Container. Defina Trailing, Leading, Top e Bottom para a Participant Collection View Cell.




A segunda visualização corresponde a View. Defina Leading e Top para a Participant Collection View Cell e altere o valor para 4.




A terceira visualização corresponde a Stack View. Defina Trailing, Leading, Top e Bottom para a Superview e altere o valor para 4.




## Permissões e temporizador de ociosidade

Retornando ao nosso `ViewController`, desabilitaremos o temporizador de ociosidade do sistema para evitar que o dispositivo entre em hibernação enquanto nossa aplicação estiver sendo utilizada:

```
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
}
```

Em seguida, solicitamos permissões de câmera e de microfone por parte do 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 da aplicação

Precisamos configurar nosso `collectionViewParticipants` com o arquivo de layout que criamos 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 cada participante, criamos uma estrutura simples chamada `StageParticipant`. Isso pode ser incluso no arquivo `ViewController.swift` ou um novo arquivo pode ser criado.

```
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 acompanhar esses participantes, mantemos uma variedade deles como propriedade privada em nosso `ViewController`:

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

Essa propriedade será usada para potencializar nosso `UICollectionViewDataSource` que foi vinculado do roteiro visual anteriormente:

```
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 sua própria visualização prévia antes de ingressar em um palco, criamos um participante local imediatamente:

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

Isso resulta na renderização de uma célula de participante imediatamente após a execução da aplicação, representando o participante local.

Os usuários querem ver a si mesmos antes de ingressar em um palco, por isso, em seguida, implementamos o método `setupLocalUser()` que foi chamado pelo código de gerenciamento de permissões anteriormente. Armazenamos a referência da câmera e do microfone como objetos `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)
}
```

Aqui encontramos a câmera e o microfone do dispositivo por meio do SDK e os armazenamos em nosso objeto `streams` local e, em seguida, atribuímos a matriz de `streams` do primeiro participante (o participante local que criamos anteriormente) aos nossos `streams`. Por fim, chamamos `participantsChanged` com um `index` de 0 e `changeType` de `updated`. Essa função é uma função auxiliar para atualizar nosso `UICollectionView` com animações bonitas. Ela será algo assim:

```
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)])
    }
}
```

Não se preocupe com `cell.set` ainda; abordaremos isso mais tarde, mas é aí que renderizaremos o conteúdo da célula com base no participante.

O `ChangeType` é uma enumeração simples:

```
enum ChangeType {
    case joined, updated, left
```

```
}
```

Por fim, desejamos acompanhar se o palco está conectado. Usamos um simples `bool` para acompanhar isso, que atualizará automaticamente nossa interface do usuário quando ela for atualizada.

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

## Implementar o SDK do palco

Existem três conceitos principais que fundamentam a funcionalidade em tempo real: palco, estratégia e renderizador. O objetivo do projeto é minimizar a quantidade de lógica do lado do cliente que é necessária para desenvolver um produto funcional.

### IVS StageStrategy

Nossa implementação de `IVSStageStrategy` é simples:

```
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
    }
```

```
}
```

Para resumir, realizaremos publicações somente se o botão de publicação estiver na posição “ligado” e, se publicarmos, usaremos os streams que coletamos anteriormente. Por fim, para esta amostra, sempre nos inscrevemos em outros participantes, recebendo seus áudios e vídeos.

IVS StageRenderer

A implementação de `IVSStageRenderer` também é bastante simples, embora, devido ao número de funções, contenha um pouco mais de código. A abordagem geral neste renderizador é atualizar nossa matriz de `participants` quando o SDK nos notifica sobre uma alteração em um participante. Existem certos cenários nos quais lidamos com os participantes locais de maneira diferente, pois decidimos gerenciá-los nós mesmos para que eles possam ver a visualização prévia da câmera antes de ingressar.

```
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 uma extensão para converter o estado da conexão em um texto amigável:

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

## Implementando uma interface de usuário personalizada CollectionViewLayout

A organização de diferentes números de participantes pode ser complexa. Você deseja que eles ocupem todo o quadro da visualização primária, mas não quer lidar com a configuração de cada participante independentemente. Para facilitar isso, abordaremos a implementação de um `UICollectionViewLayout`.

Crie outro novo arquivo, `ParticipantCollectionViewLayout.swift`, que deve abranger o `UICollectionViewLayout`. Essa classe usará outra classe chamada `StageLayoutCalculator`, que abordaremos em breve. A classe recebe os valores de quadros calculados para cada participante e, em seguida, gera os objetos `UICollectionViewLayoutAttributes` necessários.

```
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))
            }
        }
    }
}
```

A classe mais importante é a `StageLayoutCalculator.swift`. Ela foi projetada para calcular os quadros de cada participante com base no número de participantes em um layout de linha/coluna baseado em fluxo. Cada linha tem a mesma altura que as outras, mas as colunas podem ter larguras

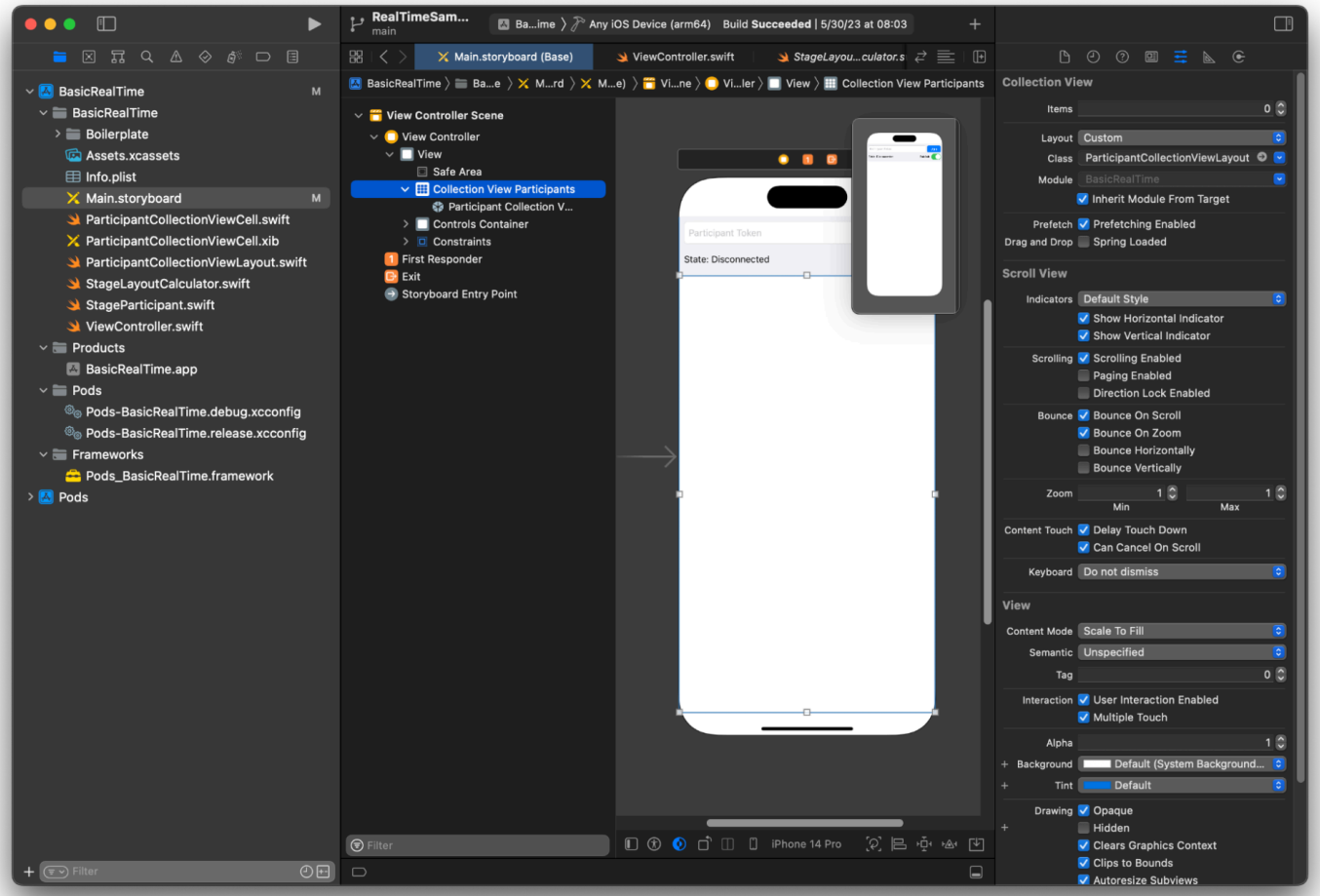
diferentes por linha. Consulte o comentário do código acima da variável `layouts` para obter uma descrição de como personalizar esse comportamento.

```
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
    }

}
```

De volta ao `Main.storyboard`, certifique-se de definir a classe de layout de `UICollectionView` para a classe que acabamos de criar:




## Conexão de ações da interface do usuário

Estamos quase finalizando, mas há algumas `IBActions` que precisamos criar.

Primeiro, abordaremos o botão Ingressar. Ele responde de forma diferente com base no valor de `connectingOrConnected`. Quando tudo já está conectado, ele apenas deixa o palco. Se houver desconexão, ele lê o texto do token `UITextField` e cria um novo `IVSStage` com esse texto. Em seguida, adicionamos nosso `ViewController` como `strategy`, `errorDelegate` e renderizador para o `IVSStage` e, finalmente, ingressamos no palco de forma assíncrona.

```
@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)")
        }
    }
}
```

A outra ação da interface do usuário que precisamos conectar é a opção de publicação:

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

## Renderização dos participantes

Por fim, precisamos renderizar os dados que recebemos do SDK na célula do participante que criamos anteriormente. Já temos a lógica do `UICollectionView` finalizada, então só precisamos implementar a API `set` em `ParticipantCollectionViewCell.swift`.

Começaremos com a adição da função `empty` e, em seguida, abordaremos ela detalhadamente:

```
func set(participant: StageParticipant) {

}
```

Primeiro, tratamos do estado fácil, do ID do participante, do estado de publicação e do estado de inscrição. Para esses, apenas atualizamos nosso `UILabels` diretamente:

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

As propriedades de texto das enumerações de publicação e inscrição vêm de extensões locais:

```
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()
        }
    }
}
```

Em seguida, atualizamos os estados de áudio e vídeo silenciados. Para obter os estados silenciados, precisamos encontrar o `IVSImageDevice` e o `IVSAudioDevice` da matriz de `streams`. Para otimizar a performance, lembraremos dos 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 fim, desejamos renderizar uma visualização prévia para o `imageDevice` e exibir as estatísticas de áudio do `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"
}
```

A última função que precisamos criar é `updatePreview()`, que adiciona uma visualização prévia do participante à nossa visualização:

```
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)
        }
    }
```

```
}
```

O código acima usa uma função auxiliar na `UIView` para facilitar a incorporação de subvisualizações:

```
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),
        ])
    }
}
```

# Monitoramento do Streaming em tempo real do Amazon IVS

## O que é uma sessão de palco?

Uma sessão de palco começa quando o primeiro participante entra em um palco e termina alguns minutos após o último participante parar de publicar no palco. As sessões de palco ajudam a depurar palcos de longa duração separando eventos e participantes em sessões de curta duração.

## Visualizar sessões de palco e participantes

### Instruções do console

1. Abra o console do Amazon IVS.

   (Também é possível acessar o console do Amazon IVS por meio do Console de Gerenciamento da AWS.)
2. No painel de navegação, selecione Palcos. (Se o painel de navegação estiver recolhido, primeiro abra-o escolhendo o ícone de hambúrguer.)
3. Escolha o palco para acessar a respectiva página de detalhes.
4. Role a página para baixo até ver a seção Sessões de palco e selecione uma sessão de palco para ver sua página de detalhes.
5. Para visualizar os participantes da sessão, role para baixo até ver a seção Participantes e selecione um participante para visualizar a página de detalhes, incluindo gráficos das métricas do Amazon CloudWatch.

## Visualizar eventos para um participante

Os eventos são enviados quando o status de um participante em um palco sofre alterações, como ingressar em um palco ou encontrar um erro ao tentar publicar em um palco. Nem todos os erros causam eventos, por exemplo, erros de rede do lado do cliente e erros de assinatura de token não são enviados como eventos. Para lidar com esses erros na aplicação do cliente, use os SDKs de Transmissão do IVS.

## Instruções do console

1. Navegue para a página de detalhes do participante conforme as instruções acima.
2. Role para baixo até ver a seção Eventos. Isso exibe uma lista ordenada dos eventos do participante. Consulte Como usar o Amazon EventBridge com o Amazon IVS para obter detalhes sobre eventos que são emitidos para os participantes.

## Instruções da CLI

Acessar eventos de sessão de palco com a AWS CLI é uma opção avançada e requer que você primeiro faça download e configure a CLI em sua máquina. Para obter mais detalhes, consulte o Guia do usuário da AWS Command Line Interface.

1. Listar sessões de palco para encontrar uma sessão de palco:

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

2. Listar participantes de uma sessão de palco para encontrar um participante:

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

3. Listar eventos para uma sessão palco um participante:

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

Veja uma resposta de exemplo para a chamada `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"
        }
    ]
}
```

# Acessar métricas do CloudWatch

Para que as métricas do CloudWatch estejam disponíveis, as seguintes versões do SDK de transmissão do IVS são necessárias: Web 1.5.0 ou posterior, Android 1.12.0 ou posterior ou iOS 1.12.0 ou posterior.

## Instruções do console do CloudWatch

1. Abra o console do CloudWatch em https://console.aws.amazon.com/cloudwatch/.
2. Na navegação lateral, expanda a lista suspensa Metrics (Métricas) e, em seguida, selecione All metrics (Todas as métricas).
3. Na guia Procurar, usando o menu suspenso sem rótulo à esquerda, selecione a sua região “inicial”, onde os seus canais foram criados. Para obter mais informações sobre regiões, consulte Solução global, controle regional. Para obter uma lista das regiões compatíveis, consulte a página do Amazon IVS na Referência geral da AWS.
4. Na parte inferior da guia Procurar, selecione o namespace IVSRealTime.
5. Execute um destes procedimentos:
   a. Na barra de pesquisa, insira o ID do recurso (parte do ARN, `arn:::ivs:stage/<resource id>`).

      Em seguida, selecione IVSRealtime > Métricas do Stage.

b. Se IVSRealTime aparecer como um serviço selecionável em Namespaces da AWS selecione essa opção. Ela estará listada se você usar o streaming em tempo real do Amazon IVS e estiver enviando métricas para o Amazon CloudWatch. (Se a opção IVSRealTime não estiver listada, você não terá nenhuma métrica do Amazon IVS.)

   Em seguida, escolha um agrupamento de dimensões, conforme desejado. As dimensões disponíveis estão listadas em Métricas do CloudWatch abaixo.

6. Escolha as métricas a serem adicionadas ao gráfico. As métricas disponíveis estão listadas em Métricas do CloudWatch abaixo.

Você também pode acessar o gráfico CloudWatch da sessão de transmissão na página de detalhes da sessão de transmissão selecionando o botão View in CloudWatch (Visualizar no CloudWatch).

### Instruções da CLI

Você também pode acessar as métricas usando a AWS CLI. Isso exige que você primeiro faça o download e configure a CLI em sua máquina. Para obter mais detalhes, consulte o Guia do usuário da Interface de Linhas de Comando da AWS.

Depois, para acessar as métricas do streaming em tempo real do Amazon IVS usando a AWS CLI:

- Em um prompt de comando, execute:

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

Para obter mais informações, consulte Como usar métricas do Amazon CloudWatch no Guia do usuário do Amazon CloudWatch.

## Métricas do CloudWatch: streaming em tempo real do IVS

O Amazon IVS fornece as seguintes métricas no namespace AWS/IVSRealTime.

Para que as métricas do CloudWatch estejam disponíveis, o Web Broadcast SDK 1.5.2 ou posterior deve ser usado.

A dimensão pode ter os seguintes valores válidos:

- A dimensão `Stage` é um ID de recurso (parte do ARN, `arn:::stage/<resource id>`).

- A dimensão `Participant` é um `participantID`.
- O `SimulcastLayer` é “alto”, “médio”, “baixo” ou “no-rid” para um `MediaType` de “vídeo” ou “desabilitado” para um `MediaType` de “áudio”. Esse valor também pode estar vazio.
- A dimensão `MediaType` é "vídeo" ou "áudio" (string).

| Métrica | Dimensão | Descrição |
|---|---|---|
| `DownloadPacketLoss` | `Stage` | Cada amostra representa a porcentagem de pacotes que foram perdidos por um determinado assinante durante o download do servidor do IVS.<br><br>Unidade: percentual<br><br>Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) da perda de pacotes durante o intervalo configurado |
| `DownloadPacketLoss` | `Stage,Participant` | Filtros `DownloadPacketLoss` por participante, para assinantes que também são publicadores. As amostras representam a porcentagem de pacotes que foram perdidos pelo assinante durante o download do servidor do IVS. As amostras são emitidas somente quando o participante também é um publicador.<br><br>Unidade: percentual<br><br>Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) de quadros abandonados durante o intervalo configurado |
| `DroppedFrames` | `Stage` | Cada exemplo representa a porcentagem de quadros que foram abandonados por um determinado assinante.<br><br>Unidade: percentual |

| Métrica | Dimensão | Descrição |
| --- | --- | --- |
| | | Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) de quadros abandonados durante o intervalo configurado |
| `DroppedFrames` | `Stage,Participant` | Filtros `DroppedFrames` por participante, para assinantes que também são publicadores. As amostras representam a porcentagem de quadros que foram abandonados entre o participante assinante e todos os publicadores no palco. As amostras são emitidas somente quando o participante também é um publicador.<br><br>Unidade: percentual<br><br>Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) de quadros abandonados durante o intervalo configurado |
| `PublishBitrate` | `Stage` | Os exemplos emitidos representam a taxa total na qual um determinado publicador está enviando dados de vídeo e de áudio (a soma em todas as camadas de transmissão simultânea).<br><br>Unidade: bits por segundo<br><br>Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) da taxa de bits durante o intervalo configurado |

| Métrica | Dimensão | Descrição |
| --- | --- | --- |
| `PublishBitrate` | `Stage, Participant, SimulcastLayer, MediaType` | Filtra `PublishBitrate` por participante, camada de transmissão simultânea e tipo de mídia. O ID da camada de transmissão simultânea é definido pelo SDK de transmissão. Quando a transmissão simultânea está desabilitada, o ID dessa camada está definido como "desabilitado". O tipo de mídia é vídeo ou áudio.<br><br>Unidade: bits por segundo<br><br>Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) da taxa de bits durante o intervalo configurado |
| `Publishers` | `Stage` | Número de participantes publicando no Stage.<br><br>Unidade: Contagem<br><br>Estatísticas válidas: médio, máximo, mínimo |
| `PublishResolution` | `Stage, Participant, SimulcastLayer, MediaType` | Número de pixels ao longo da menor largura ou altura do quadro. Por exemplo, para um quadro no formato de paisagem de 1920 x 1080, a PublishResolution é 1080. Para um quadro no formato de retrato de 720 x 1280, a PublishResolution é 720.<br><br>Unidade: Contagem<br><br>Estatísticas válidas: médio, máximo, mínimo |

| Métrica | Dimensão | Descrição |
| --- | --- | --- |
| `SubscribeBitrate` | `Stage` | Os exemplos emitidos representam a taxa total na qual um determinado assinante está recebendo dados de vídeo e áudio.<br><br>Unidade: bits por segundo<br><br>Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) da taxa de bits durante o intervalo configurado |
| `SubscribeBitrate` | `Stage,Participant,MediaType` | Filtros `SubscribeBitrate` por participante, para assinantes que também são publicadores. As amostras representam a taxa de bits na qual um determinado assinante está recebendo o referido `MediaType`. As amostras são emitidas somente enquanto o participante assinante está publicando.<br><br>Unidade: bits por segundo<br><br>Estatísticas válidas: médio, máximo, mínimo: o número médio, o número mais alto ou o número mais baixo (respectivamente) da taxa de bits durante o intervalo configurado |
| `Subscribers` | `Stage` | Número de participantes que são assinantes do Stage. Observe que os participantes que publicam e assinam ativamente são contados tanto como publicadores quanto como assinantes.<br><br>Unidade: Contagem<br><br>Estatísticas válidas: médio, máximo, mínimo |

# SDK de Transmissão do IVS (streaming em tempo real)

O SDK de Transmissão do streaming em tempo real do Amazon Interactive Video Services (IVS) é destinado aos desenvolvedores que estão criando aplicações com o Amazon IVS. Este SDK foi projetado para aproveitar a arquitetura do Amazon IVS e receberá continuamente melhorias e novos recursos, juntamente com o Amazon IVS. Como SDK de transmissão nativo, foi projetado para minimizar o impacto na performance em sua aplicação e nos dispositivos com os quais seus usuários acessam sua aplicação.

Observe que o SDK de transmissão é usado para enviar e receber vídeos, ou seja, você usa o mesmo SDK para hosts e espectadores. Nenhum SDK do reprodutor separado é necessário.

Sua aplicação pode aproveitar os principais recursos do Amazon IVS Broadcast SDK:

- Transmissões de alta qualidade: o SDK de transmissão oferece suporte a transmissões de alta qualidade. Capture vídeos usando sua câmera e codifique-os em até 720p.
- Ajustes de taxas de bits automáticos: como os usuários de smartphones são móveis, suas condições de rede podem mudar ao longo de uma transmissão. O SDK de transmissão do Amazon IVS ajusta automaticamente a taxa de bits de vídeo para acomodar as condições de rede em alteração.
- Compatível com retrato e paisagem: não importa como seus usuários seguram os dispositivos, a imagem é exibida na posição certa e dimensionada corretamente. O SDK de transmissão é compatível com os formatos de tela de retrato e paisagem. Ele gerencia automaticamente a proporção quando os usuários rodam o dispositivo para uma orientação diferente da configurada.
- Transmissões seguras: as transmissões dos usuários são criptografadas usando TLS, para que eles possam manter as transmissões seguras.
- Dispositivos de áudio externos: o Amazon IVS Broadcast SDK oferece suporte a conectores de áudio, USB e microfones externos Bluetooth SCO.

# Requisitos da plataforma

## Plataformas nativas

| Plataforma | Versões compatíveis |
| --- | --- |
| Android | Versão 9.0 e posteriores: observe que os clientes podem desenvolver com a versão 5.0, mas não poderão usar a funcionalidade de streaming em tempo real. |
| iOS | 14 e versões posteriores |

O IVS suporta no mínimo 4 versões principais do iOS e 6 versões principais do Android. Nosso suporte à versão atual pode ir além desses mínimos. Os clientes serão notificados por meio das notas de lançamento do SDK pelo menos 3 meses antes do fim do suporte para uma versão principal.

## Navegadores desktop

| Navegador | Plataformas com suporte | Versões compatíveis |
| --- | --- | --- |
| Chrome | Windows, macOS | Duas versões principais (versão anterior atual e mais recente) |
| Firefox | Windows, macOS | Duas versões principais (versão anterior atual e mais recente) |
| Borda | Windows 8.1 e posteriores | Duas versões principais (versão anterior atual e mais recente)<br><br>Exclui o Edge Legacy |
| Safari | macOS | Duas versões principais (versão anterior atual e mais recente) |

## Navegadores móveis (iOS e Android)

| Navegador | Plataformas com suporte | Versões compatíveis |
|---|---|---|
| Chrome | iOS, Android | Duas versões principais (versão anterior atual e mais recente) |
| Firefox | Android | Duas versões principais (versão anterior atual e mais recente) |
| Safari | iOS | Duas versões principais (versão anterior atual e mais recente) |

### Limitações conhecidas

- Em todos os dispositivos móveis, não recomendamos publicação/inscrição com quatro ou mais participantes ao mesmo tempo, devido a problemas com artefatos de vídeo e telas pretas. Se você precisar de mais participantes, configure a publicação e a inscrição somente de áudio.
- Não recomendamos compor um palco e transmiti-lo para um canal no Android móvel na Web, devido a considerações de desempenho e possíveis falhas. Se a funcionalidade de transmissão for necessária, integre o SDK de Transmissão do streaming em tempo real do IVS para Android.

# Visualizações da Web

O SDK de transmissão para Web não oferece suporte para visualizações da Web ou de ambientes semelhantes à Web (como TVs, consoles etc.). Para implementações móveis, consulte o Guia do SDK de transmissão do streaming em tempo real para Android e para iOS.

# Acesso ao dispositivo necessário

O SDK de transmissão necessita de acesso às câmeras e microfones do dispositivo, tanto as incorporadas no dispositivo como as conectadas por Bluetooth, USB ou conector de áudio.

# Suporte

O SDK de transmissão é aprimorado continuamente. Consulte Notas de release do Amazon IVS para ver as versões disponíveis e problemas corrigidos. Se for apropriado, antes de entrar em contato com o suporte, atualize sua versão do SDK de transmissão e veja se isso resolve seu problema.

## Versionamento

Os SDKs de transmissão do Amazon IVS usam versionamento semântico.

Para esta discussão, suponha que:

- A versão mais recente é 4.1.3.
- A versão mais recente da versão principal anterior é 3.2.4.
- A versão mais recente da versão 1.x é 1.5.6.

Novos recursos compatíveis com versões anteriores são adicionados como versões secundárias da versão mais recente. Nesse caso, o próximo conjunto de novos recursos vai ser adicionado como versão 4.2.0.

Compatíveis com versões anteriores, pequenas correções de bugs são adicionadas como lançamentos de patch da versão mais recente. Aqui, o próximo conjunto de pequenas correções de bugs vai ser adicionado como versão 4.1.4.

Compatíveis com versões anteriores, as principais correções de bugs são tratadas de forma diferente; estas são adicionadas a várias versões:

- Versão do patch da versão mais recente. Aqui, esta é a versão 4.1.4.
- Lançamento do patch da versão secundária anterior. Aqui, esta é a versão 3.2.5.
- Versão do patch da versão 1.x mais recente. Aqui, esta é a versão 1.5.7.

As principais correções de bugs são definidas pela equipe de produtos do Amazon IVS. Exemplos típicos são atualizações de segurança críticas e outras correções selecionadas necessárias para os clientes.

Observação: nos exemplos acima, versões lançadas incrementam sem ignorar nenhum número (por exemplo, de 4.1.3 para 4.1.4). Na realidade, um ou mais números de patch podem permanecer

internos e não ser liberados, de modo que a versão lançada pode ser incrementada de 4.1.3 para, digamos, 4.1.6.

# SDK de Transmissão do IVS: Guia da Web (Streaming em tempo real)

O SDK de transmissão do streaming em tempo real do IVS para Web fornece aos desenvolvedores as ferramentas necessárias para criar experiências interativas e em tempo real na Web. Esse SDK destina-se a desenvolvedores que estão criando aplicações para a Web com o Amazon IVS.

O SDK de transmissão para Web possibilita que os participantes enviem e recebam vídeos. O SDK oferece suporte para as seguintes operações:

- Entrar em um palco
- Publicar mídia para outros participantes do palco
- Inscrever-se na mídia de outros participantes do palco
- Gerenciar e monitorar vídeos e áudios publicados no palco
- Obter estatísticas WebRTC para cada conexão de pares
- Todas as operações do SDK de transmissão do streaming de baixa latência do IVS para Web

Versão mais recente do SDK de transmissão na Web: 1.8.0 (notas de lançamento)

Documentação de referência: Para obter informações sobre os métodos mais importantes disponíveis no Amazon IVS Web Broadcast SDK, consulte https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference. Verifique se a versão mais atual do SDK está selecionada.

Código de exemplo: os exemplos abaixo são um bom lugar para aprender rapidamente a usar o SDK:

- HTML e JavaScript
- React

Requisitos de plataforma: consulte SDK de Transmissão do Amazon IVS para obter uma lista das plataformas compatíveis.

# Conceitos básicos

## Importações

Os blocos de criação para tempo real estão localizados em um namespace diferente dos módulos de transmissão raiz.

Uso de uma etiqueta de script

Ao usar as mesmas importações de script, as classes e as enumerações definidas nos exemplos abaixo podem ser encontradas no objeto global `IVSBroadcastClient`:

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

Uso de npm

As classes, as enumerações e os tipos também podem ser importados do módulo do pacote:

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

## Solicitar permissões

Sua aplicação deverá solicitar permissão para acessar a câmera e o microfone do usuário, e isso deverá ser servido por meio de HTTPS. (Isso não é específico do Amazon IVS; é necessário para qualquer site que precise acessar câmeras e microfones.)

Aqui está um exemplo de função que mostra como é possível solicitar e capturar permissões para ambos os dispositivos de áudio e vídeo:

```
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 obter informações adicionais, consulte a API de permissões MediaDevices e. getUserMedia().

## Listar dispositivos disponíveis

Para ver quais dispositivos estão disponíveis para captura, consulte o método MediaDevices.enumerateDevices () do navegador:

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

## Recuperar um MediaStream de um dispositivo

Depois de adquirir a lista de dispositivos disponíveis, é possível recuperar um stream de qualquer número de dispositivos. Por exemplo, é possível usar o método `getUserMedia()` para recuperar um stream de uma câmera.

Se você quiser especificar de qual dispositivo capturar o stream, é possível definir explicitamente o `deviceId` na seção `audio` ou `video` das restrições de mídia. Como alternativa, é possível omitir o `deviceId` e fazer com que os usuários selecionem seus dispositivos no prompt do navegador.

Também é possível especificar uma resolução de câmera ideal usando as restrições `width` e `height`. (Leia mais sobre essas restrições aqui.) O SDK aplica automaticamente restrições de largura e altura que correspondem à resolução máxima de transmissão, mas é melhor você mesmo também aplicá-las para garantir que a proporção da fonte não seja alterada depois que ela for adicionada ao SDK.

Para streaming em tempo real, certifique-se de que a mídia esteja restrita à resolução de 720p. Especificamente, seus valores `getUserMedia` e de `getDisplayMedia` restrição para largura e altura não devem exceder 921600 (1280*720) quando multiplicados juntos.

```
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 },
});
```

# Publicação e inscrição

## Conceitos

Existem três conceitos principais que fundamentam a funcionalidade em tempo real: palco, estratégia e eventos. O objetivo do projeto é minimizar a quantidade de lógica do lado do cliente que é necessária para desenvolver um produto funcional.

### Estágio

A classe `Stage` corresponde ao principal ponto de interação entre a aplicação de host e o SDK. Ela representa o próprio palco e é usada para entrar e sair do palco. Criar e entrar em um palco requer uma string de token válida e não expirada do ambiente de gerenciamento (representada como `token`). Entrar e sair de um palco é simples:

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

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

stage.leave();
```

### Strategy

A interface `StageStrategy` fornece uma maneira para a aplicação de host comunicar o estado desejado do palco ao SDK. Três funções precisam ser implementadas: `shouldSubscribeToParticipant`, `shouldPublishParticipant` e `stageStreamsToPublish`. Todas serão discutidas abaixo.

Para usar uma estratégia definida, passe-a para o `Stage` de criação. Veja a seguir um exemplo completo de uma aplicação que usa uma estratégia para publicar a webcam de um participante no palco e realizar a inscrição para todos os participantes. A finalidade de cada função de estratégia necessária é explicada em detalhes nas seções subsequentes.

```
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();
```

Como se inscrever como participante

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

Quando um participante remoto entra no palco, o SDK consulta a aplicação de host sobre o estado de inscrição desejado para esse participante. As opções são `NONE`, `AUDIO_ONLY` e `AUDIO_VIDEO`. Ao retornar um valor para essa função, a aplicação de host não precisa se preocupar com o estado de publicação, o estado atual da inscrição ou o estado da conexão do palco. Se `AUDIO_VIDEO` for retornado, o SDK aguardará até que o participante remoto esteja publicando antes de inscrever e atualizará a aplicação de host ao emitir eventos durante todo o processo.

Veja a seguir uma amostra de implementação:

```
const strategy = {

   shouldSubscribeToParticipant: (participant) => {
      return SubscribeType.AUDIO_VIDEO;
```

```
   }

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

Esta é a implementação completa desta função para uma aplicação de host que sempre deseja que todos os participantes se vejam, por exemplo, uma aplicação de bate-papo por vídeo.

Implementações mais avançadas também são possíveis. Use a propriedade `userInfo` em `ParticipantInfo` para se inscrever, de forma seletiva, como participante com base nos recursos fornecidos pelo 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
}
```

Isso pode ser usado para criar um palco no qual os moderadores podem monitorar todos os convidados sem serem vistos ou ouvidos. A aplicação de host pode usar uma lógica de negócios adicional para permitir que os moderadores se vejam, mas permaneçam invisíveis para os convidados.

Publicação

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

Uma vez conectado ao palco, o SDK consulta a aplicação de host para visualizar se um determinado participante deve realizar uma publicação. Isso é invocado somente para participantes locais que têm permissão para realizar publicações com base no token fornecido.

Veja a seguir uma amostra de implementação:

```
const strategy = {

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

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

Isso é para uma aplicação de bate-papo por vídeo padrão na qual os usuários sempre desejam realizar publicações. Eles podem ativar e desativar o áudio e o vídeo para serem ocultados ou vistos/ouvidos instantaneamente. (Também é possível usar publicar/cancelar a publicação, mas isso é muito mais lento. Ativar/Desativar o áudio é preferível para casos de uso em que é desejável alterar a visibilidade com frequência.)

Como escolher streams para realizar publicações

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

Ao realizar publicações, isso é usado para determinar quais streams de áudio e de vídeo devem ser publicados. Isso será abordado com mais detalhes posteriormente em Publish a Media Stream.

Como atualizar a estratégia

A estratégia pretende ser dinâmica, ou seja, os valores retornados de qualquer uma das funções acima podem ser alterados a qualquer momento. Por exemplo, se a aplicação de host não desejar realizar publicações até que o usuário final toque em um botão, você poderá retornar uma variável de `shouldPublishParticipant` (algo como `hasUserTappedPublishButton`). Quando essa variável for alterada com base em uma interação do usuário final, chame `stage.refreshStrategy()` para sinalizar ao SDK que ele deve consultar a estratégia para obter os valores mais recentes, aplicando somente o que sofreu alterações. Se o SDK observa que o valor `shouldPublishParticipant` foi alterado, ele inicia o processo de publicação. Se as consultas do SDK e todas as funções retornarem o mesmo valor anterior, a chamada `refreshStrategy` não modificará o palco.

Se o valor de retorno de `shouldSubscribeToParticipant` for alterado de `AUDIO_VIDEO` para `AUDIO_ONLY`, a transmissão de vídeo será removida para todos os participantes com os valores retornados alterados, caso uma transmissão de vídeo tenha existido anteriormente.

Geralmente, o palco usa a estratégia para aplicar com mais eficiência a diferença entre as estratégias anteriores e atuais, sem que a aplicação de host precise se preocupar com todo o estado necessário para realizar o gerenciamento adequado. Por causa disso, pense na chamada `stage.refreshStrategy()` como uma operação barata, porque ela não faz nada a menos que a estratégia seja alterada.

Eventos

Uma instância `Stage` é um emissor de eventos. Ao usar `stage.on()`, o estado do palco é comunicado à aplicação de host. Geralmente, as atualizações na interface do usuário da aplicação de host podem ser totalmente apoiadas pelos eventos. Os eventos são os seguintes:

```
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 a maioria desses eventos, o correspondente `ParticipantInfo` é fornecido.

Não é esperado que as informações fornecidas pelos eventos impactem os valores de retorno da estratégia. Por exemplo, não se espera que o valor de retorno de `shouldSubscribeToParticipant` seja alterado quando `STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` for chamado. Se a aplicação de host desejar inscrever um determinado participante, ele deverá retornar o tipo de inscrição desejado, independentemente do estado de publicação desse participante. O SDK é responsável por garantir que o estado desejado da estratégia seja acionado no momento correto com base no estado do palco.

## Publicação de uma transmissão de mídia

Dispositivos locais, como microfones e câmeras, são recuperados usando as mesmas etapas descritas acima em Recuperar um de um MediaStream dispositivo. No exemplo, usamos `MediaStream` para criar uma lista de objetos `LocalStageStream` usados para publicação pelo SDK:

```
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
    }
}
```

## Publicação de um compartilhamento de tela

Geralmente, as aplicações precisam publicar um compartilhamento de tela além da câmera da Web do usuário. A publicação de um compartilhamento de tela exige a criação de um `Stage` adicional com o próprio token exclusivo.

```
// 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();
```

## Exibição e remoção de participantes

Após a conclusão da inscrição, você recebe uma matriz de objetos `StageStream` por meio do evento `STAGE_PARTICIPANT_STREAMS_ADDED`. O evento também fornece informações do participante para ajudar na exibição de transmissões de mídia:

```
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));
})
```

Quando um participante interrompe as publicações ou cancela a inscrição de uma transmissão, a função `STAGE_PARTICIPANT_STREAMS_REMOVED` é chamada com as transmissões que foram removidas. As aplicações de host devem usar isso como um sinal para remover a transmissão de vídeo do participante do DOM.

`STAGE_PARTICIPANT_STREAMS_REMOVED` é invocada para todos os cenários em que uma transmissão pode ser removida, incluindo:

- Um participante remoto que interrompe as publicações.
- Um dispositivo local que cancela a inscrição ou altera a inscrição de `AUDIO_VIDEO` para `AUDIO_ONLY`.
- Um participante remoto que sai do palco.

- Um participante local que sai do palco.

Como `STAGE_PARTICIPANT_STREAMS_REMOVED` é invocada para todos os cenários, nenhuma lógica de negócios personalizada é necessária para remover participantes da IU durante operações de saída remotas ou locais.

### Ativação ou desativação do áudio para transmissões de mídia

Os objetos `LocalStageStream` têm uma função `setMuted` que controla se a transmissão é silenciada. Essa função pode ser chamada na transmissão antes ou depois de ser retornada da função de estratégia `stageStreamsToPublish`.

Importante: se uma nova instância de objeto `LocalStageStream` for retornada por `stageStreamsToPublish` após uma chamada para `refreshStrategy`, o estado mudo do novo objeto de transmissão será aplicado ao palco. Tenha cuidado ao criar novas instâncias `LocalStageStream` para garantir que o estado mudo esperado seja mantido.

### Monitoramento do estado mudo da mídia do participante remoto

Quando os participantes alteram o estado mudo do vídeo ou do áudio, o evento `STAGE_STREAM_MUTE_CHANGED` é acionado com uma lista de transmissões que foram alteradas. Use a propriedade `isMuted` na `StageStream` para atualizar a IU adequadamente:

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

Além disso, você pode consultar as informações StageParticipantInfode estado sobre se o áudio ou o vídeo estão silenciados:

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

## Obtenção de estatísticas WebRTC

Para obter as estatísticas WebRTC mais recentes para uma transmissão de publicação ou de inscrição, use `getStats` em `StageStream`. Este é um método assíncrono com o qual você pode recuperar estatísticas utilizando await ou encadeando uma promessa. O resultado é um `RTCStatsReport`, que corresponde a um dicionário que contém todas as estatísticas padrão.

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

## Otimização de mídia

É recomendável limitar as chamadas `getUserMedia` e `getDisplayMedia` para as seguintes restrições com a finalidade de obter a melhor performance:

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

É possível restringir ainda mais a mídia por meio de opções adicionais passadas para o construtor `LocalStageStream`:

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

No código acima:

- `minBitrate` define uma taxa de bits mínima que o navegador deve usar. No entanto, um stream de vídeo de baixa complexidade pode impulsionar o codificador a diminuir a taxa de bits.
- `maxBitrate` define uma taxa de bits máxima que o navegador não deve exceder para este stream.
- `maxFramerate` define uma taxa de quadros máxima que o navegador não deve exceder para este stream.
- A opção `simulcast` pode ser usada somente em navegadores baseados no Chromium. Ela possibilita o envio de três camadas de representação do stream.
  - Isso permite que o servidor escolha qual representação enviar aos outros participantes, com base em suas limitações de rede.
  - Quando `simulcast` é especificada junto com um valor `maxBitrate` e/ou `maxFramerate`, espera-se que a camada de representação mais alta seja configurada considerando esses valores, desde que `maxBitrate` não seja inferior ao valor de `maxBitrate` padrão para a segunda camada mais alta do SDK interno de 900 kbps.
  - Se `maxBitrate` for especificada com um valor muito inferior em comparação com o valor padrão da segunda camada mais alta, a `simulcast` será desabilitada.
  - A `simulcast` não pode ser ativada e desativada sem republicar a mídia por meio de uma combinação de `shouldPublishParticipant` retornar `false`, chamar `refreshStrategy`, fazer `shouldPublishParticipant` retornar `true` e chamar `refreshStrategy` novamente.

## Obtenção de atributos do participante

Se você especificar atributos na solicitação de endpoint `CreateParticipantToken`, poderá visualizar os atributos nas propriedades `StageParticipantInfo`:

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

## Tratamento de problemas de rede

Quando a conexão de rede do dispositivo local é perdida, o SDK tenta se reconectar internamente sem nenhuma ação do usuário. Em alguns casos, o SDK não obtém êxito e a ação do usuário é necessária.

De maneira geral, o estado do palco pode ser tratado por meio do 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;
})
```

Em geral, encontrar erros após entrar com êxito em um palco indica que o SDK perdeu a conexão e não conseguiu restabelecê-la. Crie um novo objeto `Stage` e tente entrar quando as condições da rede melhorarem.

## Transmissão do palco para um canal do IVS

Para transmitir um palco, crie uma sessão `IVSBroadcastClient` separada e, em seguida, siga as instruções usuais para transmissão com o SDK, descritas acima. A lista de `StageStream` expostas por meio de `STAGE_PARTICIPANT_STREAMS_ADDED` pode ser usada para recuperar as transmissões de mídia participantes que podem ser aplicadas à composição do fluxo de transmissão da seguinte forma:

```
// 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;
        }
    })
})
```

Você também pode compor um palco e transmiti-lo para um canal de baixa latência do IVS para alcançar um público maior. Consulte Enabling Multiple Hosts on an Amazon IVS Stream no Guia do usuário do streaming de baixa latência do IVS.

## Problemas conhecidos e soluções

- Ao fechar as guias do navegador ou sair dos navegadores sem chamar `stage.leave()`, os usuários ainda podem aparecer na sessão com um quadro congelado ou tela preta por até dez segundos.

  Solução alternativa: nenhuma.
- As sessões do Safari aparecem intermitentemente com uma tela preta para os usuários que entram após o início de uma sessão.

  Solução alternativa: atualize o navegador e reconecte a sessão.
- O Safari não se recupera normalmente da troca de redes.

  Solução alternativa: atualize o navegador e reconecte a sessão.
- O console do desenvolvedor repete um erro `Error: UnintentionalError at StageSocket.onClose`.

  Solução alternativa: somente um palco pode ser criado por token de participante. Esse erro ocorre quando mais de uma instância `Stage` é criada com o mesmo token de participante, independentemente de a instância estar em um dispositivo ou em vários dispositivos.
- Você pode ter problemas para manter um `StageParticipantPublishState.PUBLISHED` estado e receber `StageParticipantPublishState.ATTEMPTING_PUBLISH` estados repetidos ao ouvir o `StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` evento.

Solução alternativa: restrinja a resolução do vídeo a 720p ao invocar ou. `getUserMedia` `getDisplayMedia` Especificamente, seus valores `getUserMedia` e de `getDisplayMedia` restrição para largura e altura não devem exceder 921600 (1280*720) quando multiplicados juntos.

## Limitações do Safari

- Para negar um aviso de permissões, é necessário redefinir a permissão nas configurações do site do Safari no nível do sistema operacional.
- O Safari não detecta nativamente todos os dispositivos com a mesma eficácia que o Firefox ou o Chrome. Por exemplo, a câmera virtual OBS não é detectada.

## Limitações do Firefox

- As permissões do sistema precisam estar habilitadas para que o Firefox compartilhe a tela. Depois de ativá-las, o usuário deve reiniciar o Firefox para que ele funcione corretamente; caso contrário, se as permissões forem percebidas como bloqueadas, o navegador lançará uma NotFoundErrorexceção.
- O método `getCapabilities` está ausente. Isso significa que os usuários não conseguem obter a resolução ou a proporção da faixa de mídia. Veja este tópico do bugzilla.
- Várias propriedades `AudioContext` estão ausentes; por exemplo, latência e contagem de canais. Isso pode representar um problema para usuários avançados que desejem manipular as faixas de áudio.

- Os feeds de câmera de `getUserMedia` são restritos a uma proporção de 4:3 no macOS. Veja o tópico 1 do bugzilla e o tópico 2 do bugzilla.

- Não há suporte para a captura de áudio com `getDisplayMedia`. Veja este tópico do bugzilla.
- A taxa de quadros na captura de tela está abaixo do ideal (aproximadamente a 15 fps?). Veja este tópico do bugzilla.

## Limitações da Web móvel

- getDisplayMediao compartilhamento de tela não é suportado em dispositivos móveis.

Solução alternativa: nenhuma.

- O participante leva de 15 a 30 segundos para sair ao fechar um navegador sem chamar `leave()`.

  Solução alternativa: adicione uma interface de usuário que incentive os usuários a se desconectarem adequadamente.

- A aplicação em segundo plano faz com que a publicação do vídeo pare.

  Solução alternativa: exiba uma lista de interface do usuário quando o publicador estiver pausado.

- A taxa de quadros do vídeo cai por aproximadamente 5 segundos após ativar o som de uma câmera em dispositivos Android.

  Solução alternativa: nenhuma.

- O feed de vídeo fica esticado em rotação para o iOS 16.0.

  Solução alternativa: exiba uma interface de usuário descrevendo esse problema conhecido do sistema operacional.

- A troca do dispositivo de entrada de áudio alterna automaticamente o dispositivo de saída de áudio.

  Solução alternativa: nenhuma.

- Colocar o navegador em segundo plano faz com que o fluxo de publicação fique preto e produza somente áudio.

  Solução alternativa: nenhuma. Isso é por motivos de segurança.

## Como tratar erros

Esta seção é uma visão geral das condições de erro, como o SDK de Transmissão na Web as reporta à aplicação e o que uma aplicação deve fazer quando esses erros são encontrados. Existem quatro categorias de erros:

```
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
  }
});
```

## Erros de instanciação de estágio

A instanciação de estágio não valida os tokens remotamente, mas verifica alguns problemas básicos de token que podem ser validados no lado do cliente. Como resultado, o SDK pode gerar um erro.

Token participante malformado

Isso ocorre quando o token de estágio está malformado. Ao instanciar um estágio, o SDK gera um erro com esta mensagem: “Erro ao analisar o token de estágio”.

Ação: crie um token válido e tente instanciar novamente.

## Erros de ingresso no estágio

São os erros que podem ocorrer ao tentar ingressar inicialmente em um estágio.

O estágio foi excluído

Ocorre ao ingressar em um estágio (associado a um token) que foi excluído. O método `join` SDK gera um erro com esta mensagem: “InitialConnectTimedOut após 10 segundos”.

Ação: crie um token válido com um novo estágio e tente entrar novamente.

Token de participante expirado

Ocorre quando o token expira. O método `join` do SDK gera um erro com esta mensagem: “O token expirou e não é mais válido”.

Ação: crie um novo token e tente entrar novamente.

Token participante inválido ou revogado

Ocorre quando o token não é válido ou foi revogado/desconectado. O método `join` SDK gera um erro com esta mensagem: “InitialConnectTimedOut após 10 segundos”.

Ação: crie um novo token e tente entrar novamente.

Token desconectado

Isso ocorre quando o token do estágio não está malformado, mas é rejeitado pelo servidor do Stages. O método `join` SDK gera um erro com esta mensagem: “InitialConnectTimedOut após 10 segundos”.

Ação: crie um token válido e tente entrar novamente.

Erros de rede para entrada inicial

Isso ocorre quando o SDK não consegue entrar em contato com o servidor do Stages para estabelecer uma conexão. O método `join` SDK gera um erro com esta mensagem: “InitialConnectTimedOut após 10 segundos”.

Ação: aguarde até que a conectividade do dispositivo se recupere e tente entrar novamente.

Erros de rede quando já ingressado

Se a conexão de rede do dispositivo cair, o SDK poderá perder sua conexão com os servidores de Estágios. Você pode ver erros no console porque o SDK não consegue mais acessar serviços de back-end. POSTs em https://broadcast.stats.live-video.net falharão.

Se estiver publicando e/ou assinando, você verá erros no console relacionados a tentativas de publicação/assinatura.

Internamente, o SDK tentará se reconectar com uma estratégia de recuo exponencial.

Ação: aguarde até que a conectividade do dispositivo se recupere. Se estiver publicando ou assinando, atualize a estratégia para garantir a republicação dos seus fluxos de mídia.

### Erros de publicação e assinatura

Erro de publicação: estados de publicação

O SDK relata `ERRORED` quando uma publicação falha. Pode ocorrer devido às condições da rede ou quando um estágio está lotado para editores.

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

Ação: atualize a estratégia para tentar a republicação dos seus fluxos de mídia.

Erros de assinatura

O SDK relata `ERRORED` quando uma assinatura falha. Pode ocorrer devido às condições da rede ou quando um estágio está lotado para assinantes.

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

Ação: atualize a estratégia para tentar uma nova assinatura.

## SDK de Transmissão do IVS: Guia do Android (streaming em tempo real)

O SDK de transmissão do streaming em tempo real do IVS para Android possibilita que os participantes enviem e recebam vídeos no Android.

O pacote `com.amazonaws.ivs.broadcast` implementa a interface descrita neste documento. O SDK oferece suporte para as seguintes operações:

- Entrar em um palco

- Publicar mídia para outros participantes do palco
- Inscrever-se na mídia de outros participantes do palco
- Gerenciar e monitorar vídeos e áudios publicados no palco
- Obter estatísticas WebRTC para cada conexão de pares
- Todas as operações do SDK de transmissão do streaming de baixa latência do IVS para Android

Versão mais recente do SDK de transmissão para Android: 1.14.1 (notas de versão)

Documentação de referência: Para obter informações sobre os métodos mais importantes disponíveis no SDK de transmissão do Amazon IVS para Android, consulte a documentação de referência em https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/android/.

Código de exemplo: consulte o repositório de amostras do Android em GitHub: https://github.com/aws-samples/ amazon-ivs-broadcast-android -sample.

Requisitos da plataforma: Android 9.0 e versões posteriores.

## Conceitos básicos

### Instalar a biblioteca

Para adicionar a biblioteca de transmissão do Amazon IVS para Android a seu ambiente de desenvolvimento do Android, adicione a biblioteca ao seu arquivo `build.gradle` do módulo, conforme mostrado aqui (para a versão mais recente do SDK de transmissão do Amazon IVS):

```
repositories {
    mavenCentral()
}

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

Adicione a seguinte permissão ao manifesto para permitir que o SDK habilite e desabilite o viva-voz:

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

Como alternativa, para instalar o SDK manualmente, baixe a versão mais recente neste local:

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

Certifique-se de fazer download do `aar` com `-stages` em anexo.

## Solicitar permissões

Sua aplicação deverá solicitar permissão para acessar a câmera e o microfone do usuário. (Isso não é específico do Amazon IVS; é necessário para qualquer aplicação que precise acessar câmeras e microfones.)

Aqui, verificamos se o usuário já concedeu permissões e, caso contrário, nós as solicitamos:

```
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;
    }
}
```

Aqui, recebemos a resposta do usuário:

```
@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();
    }
}
```

# Publicação e inscrição

## Conceitos

Existem três conceitos principais que fundamentam a funcionalidade em tempo real: palco, estratégia e renderizador. O objetivo do projeto é minimizar a quantidade de lógica do lado do cliente que é necessária para desenvolver um produto funcional.

### Estágio

A classe `Stage` corresponde ao principal ponto de interação entre a aplicação de host e o SDK. Ela representa o próprio palco e é usada para entrar e sair do palco. Criar e entrar em um palco requer uma string de token válida e não expirada do ambiente de gerenciamento (representada como `token`). Entrar e sair de um palco é simples.

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

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

stage.leave();
```

Na classe `Stage`, também é possível anexar o `StageRenderer`:

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

### Strategy

A interface `Stage.Strategy` fornece uma maneira para a aplicação de host comunicar o estado desejado do palco ao SDK. Três funções precisam ser implementadas: `shouldSubscribeToParticipant`, `shouldPublishFromParticipant` e `stageStreamsToPublishForParticipant`. Todas serão discutidas abaixo.

### Como se inscrever como participante

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

Quando um participante remoto entra no palco, o SDK consulta a aplicação de host sobre o estado de inscrição desejado para esse participante. As opções são `NONE`, `AUDIO_ONLY` e `AUDIO_VIDEO`. Ao retornar um valor para essa função, a aplicação de host não precisa se preocupar com o estado de publicação, o estado atual da inscrição ou o estado da conexão do palco. Se `AUDIO_VIDEO` for retornado, o SDK aguardará até que o participante remoto esteja publicando antes de inscrever e atualizará a aplicação de host por meio do renderizador durante todo o processo.

Veja a seguir uma amostra de implementação:

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

Esta é a implementação completa desta função para uma aplicação de host que sempre deseja que todos os participantes se vejam, por exemplo, uma aplicação de bate-papo por vídeo.

Implementações mais avançadas também são possíveis. Use a propriedade `userInfo` em `ParticipantInfo` para se inscrever, de forma seletiva, como participante com base nos recursos fornecidos pelo 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;
 }
}
```

Isso pode ser usado para criar um palco no qual os moderadores podem monitorar todos os convidados sem serem vistos ou ouvidos. A aplicação de host pode usar uma lógica de negócios adicional para permitir que os moderadores se vejam, mas permaneçam invisíveis para os convidados.

Publicação

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

Uma vez conectado ao palco, o SDK consulta a aplicação de host para visualizar se um determinado participante deve realizar uma publicação. Isso é invocado somente para participantes locais que têm permissão para realizar publicações com base no token fornecido.

Veja a seguir uma amostra de implementação:

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

Isso é para uma aplicação de bate-papo por vídeo padrão na qual os usuários sempre desejam realizar publicações. Eles podem ativar e desativar o áudio e o vídeo para serem ocultados ou vistos/ouvidos instantaneamente. (Também é possível usar publicar/cancelar a publicação, mas isso é muito mais lento. Ativar/Desativar o áudio é preferível para casos de uso em que é desejável alterar a visibilidade com frequência.)

Como escolher streams para realizar publicações

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

Ao realizar publicações, isso é usado para determinar quais streams de áudio e de vídeo devem ser publicados. Isso será abordado com mais detalhes posteriormente em Publish a Media Stream.

Como atualizar a estratégia

A estratégia pretende ser dinâmica, ou seja, os valores retornados de qualquer uma das funções acima podem ser alterados a qualquer momento. Por exemplo, se a aplicação de host não desejar realizar publicações até que o usuário final toque em um botão, você poderá retornar uma variável de `shouldPublishFromParticipant` (algo como `hasUserTappedPublishButton`). Quando essa variável for alterada com base em uma interação do usuário final, chame

`stage.refreshStrategy()` para sinalizar ao SDK que ele deve consultar a estratégia para obter os valores mais recentes, aplicando somente o que sofreu alterações. Se o SDK observar que o valor `shouldPublishFromParticipant` foi alterado, ele iniciará o processo de publicação. Se as consultas do SDK e todas as funções retornarem o mesmo valor anterior, a chamada `refreshStrategy` não realizará nenhuma modificação no palco.

Se o valor de retorno de `shouldSubscribeToParticipant` for alterado de `AUDIO_VIDEO` para `AUDIO_ONLY`, a transmissão de vídeo será removida para todos os participantes com os valores retornados alterados, caso uma transmissão de vídeo tenha existido anteriormente.

Geralmente, o palco usa a estratégia para aplicar com mais eficiência a diferença entre as estratégias anteriores e atuais, sem que a aplicação de host precise se preocupar com todo o estado necessário para realizar o gerenciamento adequado. Por causa disso, pense na chamada `stage.refreshStrategy()` como uma operação barata, porque ela não faz nada a menos que a estratégia seja alterada.

Renderizador

A interface `StageRenderer` comunica o estado do palco à aplicação de host. Geralmente, as atualizações na interface do usuário da aplicação de host podem ser baseadas inteiramente nos eventos fornecidos pelo renderizador. O renderizador fornece as seguintes funções:

```
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 a maioria desses métodos, os correspondentes `Stage` e `ParticipantInfo` são fornecidos.

Não é esperado que as informações fornecidas pelo renderizador impactem os valores de retorno da estratégia. Por exemplo, não se espera que o valor de retorno de `shouldSubscribeToParticipant` seja alterado quando `onParticipantPublishStateChanged` for chamado. Se a aplicação de host desejar inscrever um determinado participante, ele deverá retornar o tipo de inscrição desejado, independentemente do estado de publicação desse participante. O SDK é responsável por garantir que o estado desejado da estratégia seja acionado no momento correto com base no estado do palco.

O `StageRenderer` pode ser anexado à classe de palco:

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

Observe que somente a publicação de participantes aciona `onParticipantJoined` e, sempre que um participante interrompe as publicações ou sai da sessão de palco, `onParticipantLeft` é acionado.

## Publicação de uma transmissão de mídia

Dispositivos locais, como microfones e câmeras integrados, são descobertos por meio de `DeviceDiscovery`. Veja a seguir um exemplo de como selecionar a câmera frontal e o microfone padrão e, em seguida, retorná-los como `LocalStageStreams` para serem publicados pelo 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;
}
```

## Exibição e remoção de participantes

Após a conclusão da inscrição, você receberá uma matriz de objetos `StageStream` por meio da função on`StreamsAdded` do renderizador. É possível recuperar a visualização prévia de uma `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);
```

É possível recuperar as estatísticas de nível de áudio de uma `AudioStageStream`:

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

Quando um participante interrompe as publicações ou cancela a inscrição, a função `onStreamsRemoved` é chamada com as transmissões que foram removidas. As aplicações de host devem usar isso como um sinal para remover a transmissão de vídeo do participante da hierarquia de visualização.

`onStreamsRemoved` é invocada para todos os cenários em que uma transmissão pode ser removida, incluindo:

- Um participante remoto que interrompe as publicações.
- Um dispositivo local que cancela a inscrição ou altera a inscrição de `AUDIO_VIDEO` para `AUDIO_ONLY`.
- Um participante remoto que sai do palco.
- Um participante local que sai do palco.

Como `onStreamsRemoved` é invocada para todos os cenários, nenhuma lógica de negócios personalizada é necessária para remover participantes da IU durante operações de saída remotas ou locais.

### Ativação ou desativação do áudio para transmissões de mídia

Os objetos `LocalStageStream` têm uma função `setMuted` que controla se a transmissão é silenciada. Essa função pode ser chamada na transmissão antes ou depois de ser retornada da função de estratégia `streamsToPublishForParticipant`.

Importante: se uma nova instância de objeto `LocalStageStream` for retornada por `streamsToPublishForParticipant` após uma chamada para `refreshStrategy`, o estado mudo do novo objeto de transmissão será aplicado ao palco. Tenha cuidado ao criar novas instâncias `LocalStageStream` para garantir que o estado mudo esperado seja mantido.

### Monitoramento do estado mudo da mídia do participante remoto

Quando um participante altera o estado mudo de sua transmissão de vídeo ou áudio, a função `onStreamMutedChanged` do renderizador é invocada com uma lista de transmissões que foram alteradas. Use o método `getMuted` na `StageStream` para atualizar a IU adequadamente.

```
@Override
void onStreamsMutedChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull List<StageStream> streams) {
 for (StageStream stream : streams) {
```

```
  boolean muted = stream.getMuted();
  // handle UI changes
 }
}
```

## Obtenção de estatísticas WebRTC

Para obter as estatísticas WebRTC mais recentes para uma transmissão de publicação ou uma transmissão de inscrição, use `requestRTCStats` em `StageStream`. Quando uma coleta for concluída, você receberá as estatísticas por meio do `StageStream.Listener`, que pode ser definido em `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());
  }
 }
}
```

## Obtenção de atributos do participante

Se você especificar atributos na solicitação de endpoint `CreateParticipantToken`, poderá visualizar os atributos nas propriedades `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());
 }
}
```

## Continuação da sessão em segundo plano

Quando a aplicação entra em segundo plano, você pode desejar interromper as publicações ou se inscrever somente para ouvir o áudio de outros participantes remotos. Para fazer isso, atualize a

implementação de sua `Strategy` para interromper as publicações e se inscreva como `AUDIO_ONLY` (ou `NONE`, se aplicável).

```
// 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();
}
```

## Habilitação ou desabilitação da codificação em camadas com a transmissão simultânea

Ao publicar um stream de mídia, o SDK transmite streams de vídeo de alta e de baixa qualidade, para que os participantes remotos possam se inscrever no streaming, mesmo que tenham largura de banda de downlink limitada. Por padrão, a codificação em camadas com a transmissão simultânea está ativada. É possível desativá-la ao usar a classe `StageVideoConfiguration.Simulcast`:

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

```
ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

## Limitações de configuração de vídeo

O SDK não oferece suporte para impor o modo retrato ou paisagem usando `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)`. Na orientação retrato, a dimensão menor é usada como largura, enquanto que, na orientação paisagem, essa dimensão é usada como altura. Isso significa que as seguintes duas chamadas para `setSize` têm o mesmo efeito na configuração de vídeo:

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

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

## Tratamento de problemas de rede

Quando a conexão de rede do dispositivo local é perdida, o SDK tenta se reconectar internamente sem nenhuma ação do usuário. Em alguns casos, o SDK não obtém êxito e a ação do usuário é necessária. Existem dois erros principais relacionados à perda da conexão de rede:

- Código de erro 1400, mensagem: “PeerConnection foi perdido devido a um erro de rede desconhecido”
- Código de erro 1300 com a mensagem: “Tentativas de repetição esgotadas”.

Se o primeiro erro for recebido, mas o segundo não, o SDK ainda estará conectado ao palco e tentará restabelecer as conexões automaticamente. Como proteção, você pode chamar `refreshStrategy` sem nenhuma alteração nos valores de retorno do método de estratégia para acionar uma tentativa de reconexão manual.

Se o segundo erro for recebido, as tentativas de reconexão do SDK falharam e o dispositivo local não está mais conectado ao palco. Nesse caso, tente entrar novamente no palco ao chamar `join` depois que sua conexão de rede for restabelecida.

Em geral, encontrar erros após entrar em um palco com êxito indica que o SDK não conseguiu restabelecer uma conexão. Crie um novo objeto `Stage` e tente entrar quando as condições da rede melhorarem.

### Uso de microfones Bluetooth

Para publicar usando microfones Bluetooth, você deve iniciar uma conexão por Bluetooth SCO:

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

## Problemas conhecidos e soluções

- Quando um dispositivo Android entra e sai do modo de suspensão, é possível que a visualização prévia fique em um estado de congelamento.

  Solução alternativa: crie e use um novo `Stage`.
- Quando um participante entra com um token que está sendo usado por outro participante, a primeira conexão é desconectada sem um erro específico.

  Solução alternativa: nenhuma.
- Há um problema raro em que o publicador está publicando, mas o estado de publicação que os inscritos recebem é `inactive`.

  Solução alternativa: tente sair e, em seguida, entrar novamente na sessão. Se o problema persistir, crie um novo token para o publicador.
- Um problema raro de distorção de áudio pode ocorrer intermitentemente durante uma sessão de palco, geralmente em chamadas com maior duração.

  Solução alternativa: o participante com áudio distorcido pode sair e entrar novamente na sessão ou cancelar a publicação e republicar o áudio para corrigir o problema.
- Não há suporte para microfones externos ao publicar em um palco.

  Solução alternativa: não use um microfone externo conectado por meio de USB para realizar publicações em um palco.
- Não há suporte para publicação em um palco com compartilhamento de tela usando `createSystemCaptureSources`.

  Solução alternativa: gerencie a captura do sistema manualmente usando fontes de entrada de imagem e fontes de entrada de áudio personalizadas.

- Quando uma `ImagePreviewView` é removida de uma visualização principal (por exemplo, `removeView()` é chamada na visualização principal), a `ImagePreviewView` é liberada imediatamente. A `ImagePreviewView` não apresenta nenhum quadro quando é adicionada a outra visualização principal.

  Solução alternativa: solicite outra visualização prévia usando `getPreview`.
- Ao entrar em um palco com um Samsung Galaxy S22/+ que tem o Android 12, você pode encontrar um erro 1401 e o dispositivo local pode falhar ao entrar no palco ou entrar, mas não terá o áudio.

  Solução alternativa: atualize para o Android 13.
- Ao entrar em um palco com um Nokia X20 que tem o Android 13, a câmera pode falhar ao abrir e uma exceção ser aberta.

  Solução alternativa: nenhuma.
- Dispositivos com o chipset MediaTek Helio podem não renderizar adequadamente o vídeo de participantes remotos.

  Solução alternativa: nenhuma.
- Em alguns dispositivos, o sistema operacional do dispositivo pode escolher um microfone diferente daquele selecionado por meio do SDK. Isso ocorre porque o SDK de Transmissão do Amazon IVS não pode controlar como a rota de áudio `VOICE_COMMUNICATION` é definida, pois ela varia de acordo com diferentes fabricantes de dispositivos.

  Solução alternativa: nenhuma.
- Alguns codificadores de vídeo Android não podem ser configurados com um tamanho de vídeo menor que 176x176. Configurar um tamanho menor causa um erro e impede a transmissão.

  Solução alternativa: não configure o tamanho do vídeo para ser menor que 176x176.

# Como tratar erros

## Erros fatais x Erros não fatais

O objeto de erro tem um campo booleano “is fatal” de `BroadcastException`.

Em geral, erros fatais estão relacionados à conexão com o servidor do Stages (ou uma conexão não pode ser estabelecida ou foi perdida e não pode ser recuperada). O aplicativo deve recriar o

estágios e se associar novamente, possivelmente com um novo token ou quando a conectividade do dispositivo se recuperar.

Erros não fatais geralmente estão relacionados ao estado de publicação/assinatura e são tratados pelo SDK, que repete a operação de publicação/assinatura.

Você pode verificar essa propriedade:

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

## Erros de entrada

Token malformado

Isso acontece quando o token de estágio está malformado.

O SDK gera uma exceção Java de uma chamada para `stage.join`, com o código de erro = 1000 e fatal = true.

Ação: crie um token válido e tente entrar novamente.

Token expirado

Isso acontece quando o token de estágio expirou.

O SDK gera uma exceção Java de uma chamada para `stage.join`, com o código de erro = 1001 e fatal = true.

Ação: crie um novo token e tente entrar novamente.

Token inválido ou revogado

Isso acontece quando o token do estágio não está malformado, mas é rejeitado pelo servidor do Stages. Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK chama `onConnectionStateChanged` com uma exceção, com o código de erro = 1026 e fatal = true.

Ação: crie um token válido e tente entrar novamente.

Erros de rede para entrada inicial

Isso acontece quando o SDK não consegue entrar em contato com o servidor do Stages para estabelecer uma conexão. Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK chama `onConnectionStateChanged` com uma exceção, com o código de erro = 1300 e fatal = true.

Ação: aguarde até que a conectividade do dispositivo se recupere e tente entrar novamente.

Erros de rede quando já ingressado

Se a conexão de rede do dispositivo cair, o SDK poderá perder sua conexão com os servidores de Estágios. Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK chama `onConnectionStateChanged` com uma exceção, com o código de erro = 1300 e fatal = true.

Ação: aguarde até que a conectividade do dispositivo se recupere e tente entrar novamente.

## Erros de publicação/assinatura

Inicial

Há vários erros:

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

Estes são relatados de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK repete a operação por um número limitado de vezes. as novas tentativas, o estado de publicação/assinatura é `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Se as tentativas de repetição forem bem-sucedidas, o estado mudará para `PUBLISHED` / `SUBSCRIBED`.

O SDK chama `onError` com o código de erro relevante e fatal = false.

Ação: nenhuma ação é necessária, pois o SDK tenta novamente de forma automática. Opcionalmente, a aplicação pode atualizar a estratégia para forçar mais tentativas.

Já estabelecido, em seguida reprovar

Uma publicação ou assinatura pode falhar depois de ser estabelecida, provavelmente devido a um erro de rede. O código de erro para “conexão de peer perdida devido a um erro de rede” é 1400.

Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK tenta novamente a operação de publicação/assinatura. as novas tentativas, o estado de publicação/assinatura é `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Se as tentativas de repetição forem bem-sucedidas, o estado mudará para `PUBLISHED` / `SUBSCRIBED`.

O SDK chama `onError` com o código de erro = 1400 e fatal = false.

Ação: nenhuma ação é necessária, pois o SDK tenta novamente de forma automática. Opcionalmente, a aplicação pode atualizar a estratégia para forçar mais tentativas. Se houver perda total de conectividade, é provável que a conexão com o Stages também falhe.

# SDK de Transmissão do IVS: Guia do iOS (streaming em tempo real)

O SDK de transmissão do streaming em tempo real do IVS para iOS possibilita que os participantes enviem e recebam vídeos no iOS.

O módulo `AmazonIVSBroadcast` implementa a interface descrita neste documento. Há suporte para as seguintes operações:

- Entrar em um palco
- Publicar mídia para outros participantes do palco
- Inscrever-se na mídia de outros participantes do palco
- Gerenciar e monitorar vídeos e áudios publicados no palco

- Obter estatísticas WebRTC para cada conexão de pares
- Todas as operações do SDK de transmissão do streaming de baixa latência do IVS para iOS

Versão mais recente do SDK de transmissão para iOS: 1.14.1 (notas de lançamento)

Documentação de referência: Para obter informações sobre os métodos mais importantes disponíveis no SDK de transmissão do Amazon IVS para iOS, consulte a documentação de referência em https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios/.

Código de exemplo: consulte o repositório de amostras do iOS em GitHub: https://github.com/aws-samples/ amazon-ivs-broadcast-ios -sample.

Requisitos da plataforma: iOS 14 ou versões posteriores.

## Conceitos básicos

### Instalar a biblioteca

Recomendamos que você integre o SDK de transmissão via CocoaPods. (Se preferir, você pode adicionar manualmente a estrutura do framework a seu projeto.)

Recomendado: integrar o Broadcast SDK () CocoaPods

A funcionalidade em tempo real é publicada como uma subespecificação do SDK de transmissão do streaming de baixa latência para iOS. Isso ocorre para que os clientes possam optar por incluí-la ou excluí-la com base em suas necessidades de recursos. Incluí-la aumenta o tamanho do pacote.

Os lançamentos são publicados CocoaPods sob o nome`AmazonIVSBroadcast`. Adicione esta dependência ao seu Podfile:

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

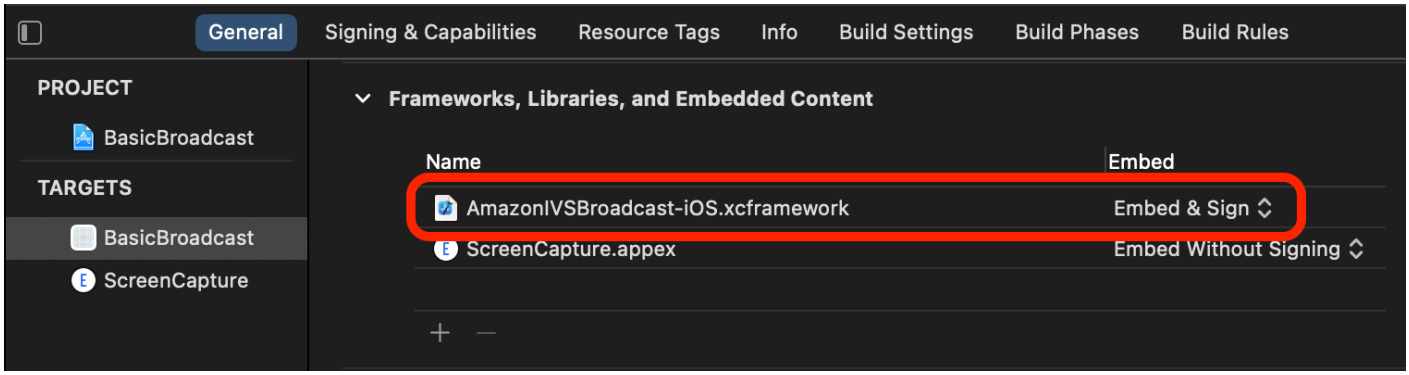
A execução do `pod install` e do SDK estará disponível em seu `.xcworkspace`.

Importante: o SDK de transmissão do streaming em tempo real do IVS (ou seja, com a subespecificação de palco) inclui todos os recursos do SDK de transmissão do streaming de baixa latência do IVS. Não é possível integrar os dois SDKs no mesmo projeto. Se você adicionar a subespecificação de estágio via CocoaPods ao seu projeto, certifique-se de remover todas as outras linhas contidas no Podfile. `AmazonIVSBroadcast` Por exemplo, certifique-se de não ter essas duas linhas em seu Podfile:

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

Abordagem alternativa: instalar o framework manualmente

1. Baixe a versão mais recente em https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Extraia o conteúdo do arquivo. `AmazonIVSBroadcast.xcframework` contém o SDK para dispositivo e para o simulador.
3. Incorporado o `AmazonIVSBroadcast.xcframework` arrastando-o para a seção Frameworks, Libraries, and Embedded Content (Frameworks, bibliotecas e conteúdo incorporado) da guia General (Geral) para o destino de sua aplicação.




## Solicitar permissões

Sua aplicação deverá solicitar permissão para acessar a câmera e o microfone do usuário. (Isso não é específico do Amazon IVS; é necessário para qualquer aplicação que precise acessar câmeras e microfones.)

Aqui, verificamos se o usuário já concedeu permissões; caso contrário, nós as solicitamos:

```
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.
}
```

É necessário fazer isso para os tipos de mídia `.video` e `.audio`, se você quiser acesso a câmeras e microfones, respectivamente.

Também é necessário adicionar entradas para `NSCameraUsageDescription` e `NSMicrophoneUsageDescription` no `Info.plist`. Caso contrário, sua aplicação falhará ao tentar solicitar permissões.

### Desativar o temporizador de ociosidade da aplicação

Isso é opcional, porém é recomendado. Isso impede que seu dispositivo entre em modo de suspensão enquanto usa o SDK de transmissão, o que interromperia a transmissão.

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

## Publicação e inscrição

### Conceitos

Existem três conceitos principais que fundamentam a funcionalidade em tempo real: palco, estratégia e renderizador. O objetivo do projeto é minimizar a quantidade de lógica do lado do cliente que é necessária para desenvolver um produto funcional.

#### Estágio

A classe `IVSStage` corresponde ao principal ponto de interação entre a aplicação de host e o SDK. A classe representa o próprio palco e é usada para entrar e sair do palco. Criar ou entrar em um palco requer uma string de token válida e não expirada do ambiente de gerenciamento (representada como `token`). Entrar e sair de um palco é simples.

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

try stage.join()
```

```
stage.leave()
```

Na classe `IVSStage`, também é possível anexar `IVSStageRenderer` e `IVSErrorDelegate`:

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

Strategy

O protocolo `IVSStageStrategy` fornece uma maneira para a aplicação de host comunicar o estado desejado do palco ao SDK. Três funções precisam ser implementadas: `shouldSubscribeToParticipant`, `shouldPublishParticipant` e `streamsToPublishForParticipant`. Todas serão discutidas abaixo.

Como se inscrever como participante

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

Quando um participante remoto entra em um palco, o SDK consulta a aplicação de host sobre o estado de inscrição desejado para esse participante. As opções são `.none`, `.audioOnly` e `.audioVideo`. Ao retornar um valor para essa função, a aplicação de host não precisa se preocupar com o estado de publicação, o estado atual da inscrição ou o estado da conexão do palco. Se `.audioVideo` for retornado, o SDK aguardará até que o participante remoto esteja publicando antes de inscrever e atualizará a aplicação de host por meio do renderizador durante todo o processo.

Veja a seguir uma amostra de implementação:

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

Esta é a implementação completa desta função para uma aplicação de host que sempre deseja que todos os participantes se vejam, por exemplo, uma aplicação de bate-papo por vídeo.

Implementações mais avançadas também são possíveis. Use a propriedade `attributes` em `IVSParticipantInfo` para se inscrever, de forma seletiva, como participante com base nos recursos fornecidos pelo servidor:

```
func stage(_ stage: IVSStage, shouldSubscribeToParticipant participant:
 IVSParticipantInfo) -> IVSStageSubscribeType {
    switch participant.attributes["role"] {
    case "moderator": return .none
    case "guest": return .audioVideo
    default: return .none
    }
}
```

Isso pode ser usado para criar um palco no qual os moderadores podem monitorar todos os convidados sem serem vistos ou ouvidos. A aplicação de host pode usar uma lógica de negócios adicional para permitir que os moderadores se vejam, mas permaneçam invisíveis para os convidados.

Publicação

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool
```

Uma vez conectado ao palco, o SDK consulta a aplicação de host para visualizar se um determinado participante deve realizar uma publicação. Isso é invocado somente para participantes locais que têm permissão para realizar publicações com base no token fornecido.

Veja a seguir uma amostra de implementação:

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool {
    return true
}
```

Isso é para uma aplicação de bate-papo por vídeo padrão na qual os usuários sempre desejam realizar publicações. Eles podem ativar e desativar o áudio e o vídeo para serem ocultados ou vistos/ouvidos instantaneamente. (Também é possível usar publicar/cancelar a publicação, mas isso é muito mais lento. Ativar/Desativar o áudio é preferível para casos de uso em que é desejável alterar a visibilidade com frequência.)

Como escolher streams para realizar publicações

```
func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream]
```

Ao realizar publicações, isso é usado para determinar quais streams de áudio e de vídeo devem ser publicados. Isso será abordado com mais detalhes posteriormente em Publish a Media Stream.

Como atualizar a estratégia

A estratégia pretende ser dinâmica, ou seja, os valores retornados de qualquer uma das funções acima podem ser alterados a qualquer momento. Por exemplo, se a aplicação de host não desejar realizar publicações até que o usuário final toque em um botão, você poderá retornar uma variável de `shouldPublishParticipant` (algo como `hasUserTappedPublishButton`). Quando essa variável for alterada com base em uma interação do usuário final, chame `stage.refreshStrategy()` para sinalizar ao SDK que ele deve consultar a estratégia para obter os valores mais recentes, aplicando somente o que sofreu alterações. Se o SDK observar que o valor `shouldPublishParticipant` foi alterado, ele iniciará o processo de publicação. Se as consultas do SDK e todas as funções retornarem o mesmo valor anterior, a chamada `refreshStrategy` não fará nenhuma modificação no palco.

Se o valor de retorno de `shouldSubscribeToParticipant` for alterado de `.audioVideo` para `.audioOnly`, a transmissão de vídeo será removida para todos os participantes com os valores retornados alterados, caso uma transmissão de vídeo tenha existido anteriormente.

Geralmente, o palco usa a estratégia para aplicar com mais eficiência a diferença entre as estratégias anteriores e atuais, sem que a aplicação de host precise se preocupar com todo o estado necessário para realizar o gerenciamento adequado. Por causa disso, pense na chamada `stage.refreshStrategy()` como uma operação barata, porque ela não faz nada a menos que a estratégia seja alterada.

Renderizador

O protocolo `IVSStageRenderer` comunica o estado do palco à aplicação de host. Geralmente, as atualizações na interface do usuário da aplicação de host podem ser baseadas inteiramente nos eventos fornecidos pelo renderizador. O renderizador fornece as seguintes funções:

```
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?)
```

Não é esperado que as informações fornecidas pelo renderizador impactem os valores de retorno da estratégia. Por exemplo, não se espera que o valor de retorno de `shouldSubscribeToParticipant` seja alterado quando `participant:didChangePublishState` for chamado. Se a aplicação de host desejar inscrever um determinado participante, ele deverá retornar o tipo de inscrição desejado, independentemente do estado de publicação desse participante. O SDK é responsável por garantir que o estado desejado da estratégia seja acionado no momento correto com base no estado do palco.

Observe que somente a publicação de participantes aciona `participantDidJoin` e, sempre que um participante interrompe as publicações ou sai da sessão de palco, `participantDidLeave` é acionado.

## Publicação de uma transmissão de mídia

Dispositivos locais, como microfones e câmeras integrados, são descobertos por meio de `IVSDeviceDiscovery`. Veja a seguir um exemplo de como selecionar a câmera frontal e o microfone padrão e, em seguida, retorná-los como `IVSLocalStageStreams` para serem publicados pelo 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]
}
```

## Exibição e remoção de participantes

Após a conclusão da inscrição, você receberá uma matriz de objetos `IVSStageStream` por meio da função `didAddStreams` do renderizador. Para visualizar previamente ou receber estatísticas de nível de áudio sobre este participante, é possível acessar o objeto `IVSDevice` subjacente da transmissão:

```
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 */
    })
}
```

Quando um participante interrompe as publicações ou cancela a inscrição, a função `didRemoveStreams` é chamada com as transmissões que foram removidas. As aplicações de host devem usar isso como um sinal para remover a transmissão de vídeo do participante da hierarquia de visualização.

`didRemoveStreams` é invocada para todos os cenários em que uma transmissão pode ser removida, incluindo:

- Um participante remoto que interrompe as publicações.
- Um dispositivo local que cancela a inscrição ou altera a inscrição de `.audioVideo` para `.audioOnly`.
- Um participante remoto que sai do palco.
- Um participante local que sai do palco.

Como `didRemoveStreams` é invocada para todos os cenários, nenhuma lógica de negócios personalizada é necessária para remover participantes da IU durante operações de saída remotas ou locais.

## Ativação ou desativação do áudio para transmissões de mídia

Os objetos `IVSLocalStageStream` têm uma função `setMuted` que controla se a transmissão é silenciada. Essa função pode ser chamada na transmissão antes ou depois de ser retornada da função de estratégia `streamsToPublishForParticipant`.

Importante: se uma nova instância de objeto `IVSLocalStageStream` for retornada por `streamsToPublishForParticipant` após uma chamada para `refreshStrategy`, o estado mudo do novo objeto de transmissão será aplicado ao palco. Tenha cuidado ao criar novas instâncias `IVSLocalStageStream` para garantir que o estado mudo esperado seja mantido.

## Monitoramento do estado mudo da mídia do participante remoto

Quando um participante altera o estado mudo de sua transmissão de vídeo ou áudio, a função `didChangeMutedStreams` do renderizador é invocada com uma matriz de transmissões que foram alteradas. Use a propriedade `isMuted` na `IVSStageStream` para atualizar a IU adequadamente:

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, didChangeMutedStreams
 streams: [IVSStageStream]) {
    streams.forEach { stream in
        /* stream.isMuted */
    }
}
```

## Criação de uma configuração de palco

Para personalizar os valores da configuração de vídeo de um palco, use `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
```

## Obtenção de estatísticas WebRTC

Para obter as estatísticas WebRTC mais recentes para uma transmissão de publicação ou uma transmissão de inscrição, use `requestRTCStats` em `IVSStageStream`. Quando uma coleta for concluída, você receberá as estatísticas por meio do `IVSStageStreamDelegate`, que pode ser definido em `IVSStageStream`. Para coletar estatísticas WebRTC de forma contínua, chame esta função em um `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)")
      }
   }
}
```

## Obtenção de atributos do participante

Se você especificar atributos na solicitação de endpoint `CreateParticipantToken`, poderá visualizar os atributos nas propriedades `IVSParticipantInfo`:

```
func stage(_ stage: IVSStage, participantDidJoin participant: IVSParticipantInfo) {
    print("ID: \(participant.participantId)")
    for attribute in participant.attributes {
        print("attribute: \(attribute.key)=\(attribute.value)")
    }
}
```

## Continuação da sessão em segundo plano

Quando a aplicação entra em segundo plano, é possível continuar no palco enquanto ouve o áudio remoto, embora não seja possível continuar enviando a própria imagem e áudio. Você precisará atualizar a implementação de sua `IVSStrategy` para interromper as publicações e se inscrever como `.audioOnly` (ou `.none`, se aplicável):

```
func stage(_ stage: IVSStage, shouldPublishParticipant participant: IVSParticipantInfo)
 -> Bool {
    return false
}
```

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

Em seguida, realize uma chamada para `stage.refreshStrategy()`.

### Habilitação ou desabilitação da codificação em camadas com a transmissão simultânea

Ao publicar um stream de mídia, o SDK transmite streams de vídeo de alta e de baixa qualidade, para que os participantes remotos possam se inscrever no streaming, mesmo que tenham largura de banda de downlink limitada. Por padrão, a codificação em camadas com a transmissão simultânea está ativada. É possível desabilitá-la com `IVSSimulcastConfiguration`:

```
// Disable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = false

let cameraStream = IVSLocalStageStream(device: camera, configuration: config)

// Other Stage implementation code
```

### Transmissão do palco para um canal do IVS

Para transmitir um palco, crie uma `IVSBroadcastSession` separada e, em seguida, siga as instruções usuais para uma transmissão com o SDK, descritas acima. A propriedade `device` na `IVSStageStream` será um `IVSImageDevice` ou um `IVSAudioDevice`, conforme mostrado no trecho de código acima. Eles podem ser conectados ao `IVSBroadcastSession.mixer` para transmitir todo o palco em um layout personalizável.

Você também pode compor um palco e transmiti-lo para um canal de baixa latência do IVS para alcançar um público maior. Consulte Enabling Multiple Hosts on an Amazon IVS Stream no Guia do usuário do streaming de baixa latência do IVS.

## Como o iOS escolhe a resolução e a taxa de quadros da câmera

A câmera gerenciada pelo SDK de transmissão otimiza sua resolução e taxa de quadros (frames-per-secondou FPS) para minimizar a produção de calor e o consumo de energia. Esta seção explica

como acontece a seleção da resolução e da taxa de quadros para ajudar aplicações no host a otimizarem seus casos de uso.

Ao criar um `IVSLocalStageStream` com um `IVSCamera`, a câmera é otimizada para uma taxa de quadros de `IVSLocalStageStreamVideoConfiguration.targetFramerate` e uma resolução de `IVSLocalStageStreamVideoConfiguration.size`. A chamada de `IVSLocalStageStream.setConfiguration` atualiza a câmera com valores mais novos.

### Prévia da câmera

Se você criar uma prévia de um `IVSCamera` sem anexá-lo a um `IVSBroadcastSession` ou `IVSStage`, o padrão será uma resolução de 1080p e uma taxa de quadros de 60 FPS.

### Transmissão de um palco

Ao usar um `IVSBroadcastSession` para transmitir um `IVSStage`, o SDK tenta otimizar a câmera com uma resolução e uma taxa de quadros que atendam aos critérios de ambas as sessões.

Por exemplo, se a configuração de transmissão estiver definida para ter uma taxa de quadros de 15 FPS e uma resolução de 1080p, enquanto o palco tiver uma taxa de quadros de 30 FPS e uma resolução de 720p, o SDK selecionará uma configuração de câmera com uma taxa de quadros de 30 FPS e uma resolução de 1080p. O `IVSBroadcastSession` eliminará todos os outros quadros da câmera e o `IVSStage` ajustará a escala da imagem de 1080p para 720p.

Se uma aplicação do host planeja usar ambos `IVSBroadcastSession` e `IVSStage` em conjunto com uma câmera, recomendamos que as propriedades `targetFramerate` e `size` das respectivas configurações correspondam. Uma incompatibilidade pode fazer com que a câmera se reconfigure durante a captura de vídeo, o que causará um breve atraso na entrega da amostra de vídeo.

Se a presença de valores idênticos não corresponder ao caso de uso da aplicação do host, criar primeiro a câmera de maior qualidade evitará que a câmera se reconfigure quando a sessão de qualidade inferior for adicionada. Por exemplo, se você transmitir em 1080p e 30 FPS e depois entrar em um palco definido para 720p e 30 FPS, a câmera não se reconfigurará e o vídeo continuará sem interrupções. Isso ocorre porque 720p é menor ou igual a 1080p e 30 FPS é menor ou igual a 30 FPS.

### Taxas de quadros, resoluções e taxas de proporções arbitrárias

A maioria dos hardwares de câmera é capaz de corresponder exatamente aos formatos comuns, como 720p a 30 FPS ou 1080p a 60 FPS. Contudo, não é possível corresponder exatamente a todos

os formatos. O SDK de transmissão escolhe a configuração da câmera com base nas seguintes regras (em ordem de prioridade):

1. A largura e a altura da resolução são maiores ou iguais à resolução desejada, mas dentro dessa restrição, a largura e a altura são as menores possíveis.
2. A taxa de quadros é maior ou igual à taxa de quadros desejada, mas dentro dessa restrição, a taxa de quadros é a mais baixa possível.
3. A taxa de proporção corresponde à proporção desejada.
4. Se houver vários formatos correspondentes, o formato com o maior campo de visão será usado.

Veja dois exemplos a seguir:

- A aplicação do host está tentando transmitir em 4K a 120 FPS. A câmera selecionada é compatível somente com 4K a 60 FPS ou 1080p a 120 FPS. O formato selecionado será 4k a 60 FPS, porque a regra de resolução tem prioridade superior em relação à regra de taxa de quadros.
- Uma resolução irregular é solicitada, 1910x1070. A câmera usará 1920x1080. Cuidado: escolher uma resolução como 1921x1080 fará com que a câmera aumente para a próxima resolução disponível (como 2592x1944), o que acarretará em penalidade na largura de banda da CPU e da memória.

### E quanto ao Android?

O Android não ajusta sua resolução ou taxa de quadros em tempo real, como o iOS, portanto, isso não afeta o SDK de Transmissão do Android.

## Problemas conhecidos e soluções

- A alteração de rotas de áudio Bluetooth pode ser imprevisível. Se você conectar um novo dispositivo no meio da sessão, o iOS poderá ou não alterar automaticamente a rota de entrada. Além disso, não é possível escolher entre vários fones de ouvido Bluetooth conectados ao mesmo tempo. Isso acontece tanto na transmissão regular quanto nas sessões de palco.

  Solução alternativa: se você planeja usar um fone de ouvido Bluetooth, conecte-o antes de iniciar a transmissão ou o palco e deixe-o conectado durante toda a sessão.
- Os participantes que usam um iPhone 14, iPhone 14 Plus, iPhone 14 Pro ou iPhone 14 Pro Max podem causar um problema de eco de áudio para os outros participantes.

Solução alternativa: os participantes que usam os dispositivos afetados podem usar fones de ouvido para evitar o problema de eco para outros participantes.

- Quando um participante entra com um token que está sendo usado por outro participante, a primeira conexão é desconectada sem um erro específico.

  Solução alternativa: nenhuma.

- Há um problema raro em que o publicador está publicando, mas o estado de publicação que os inscritos recebem é `inactive`.

  Solução alternativa: tente sair e, em seguida, entrar novamente na sessão. Se o problema persistir, crie um novo token para o publicador.

- Quando um participante está publicando ou se inscrevendo, é possível receber um erro com o código 1400 que indica desconexão devido a um problema de rede, mesmo quando a rede está estável.

  Solução alternativa: tente publicar ou se inscrever novamente.

- Um problema raro de distorção de áudio pode ocorrer intermitentemente durante uma sessão de palco, geralmente em chamadas com maior duração.

  Solução alternativa: o participante com áudio distorcido pode sair e entrar novamente na sessão ou cancelar a publicação e republicar o áudio para corrigir o problema.

# Como tratar erros

## Erros fatais x Erros não fatais

O objeto de erro tem um booleano “is fatal”. É uma entrada do dicionário em `IVSBroadcastErrorIsFatalKey` que contém um booleano.

Em geral, erros fatais estão relacionados à conexão com o servidor do Stages (ou uma conexão não pode ser estabelecida ou foi perdida e não pode ser recuperada). O aplicativo deve recriar o estágios e se associar novamente, possivelmente com um novo token ou quando a conectividade do dispositivo se recuperar.

Erros não fatais geralmente estão relacionados ao estado de publicação/assinatura e são tratados pelo SDK, que repete a operação de publicação/assinatura.

Você pode verificar essa propriedade:

```
let nsError = error as NSError
if nsError.userInfo[IVSBroadcastErrorIsFatalKey] as? Bool == true {
  // the error is fatal
}
```

## Erros de entrada

Token malformado

Isso acontece quando o token de estágio está malformado.

O SDK gera uma exceção Swift com código de erro = 1000 e IVS BroadcastErrorIsFatalKey = YES.

Ação: crie um token válido e tente entrar novamente.

Token expirado

Isso acontece quando o token de estágio expirou.

O SDK gera uma exceção Swift com código de erro = 1001 e BroadcastErrorIsFatalKey IVS = YES.

Ação: crie um novo token e tente entrar novamente.

Token inválido ou revogado

Isso acontece quando o token do estágio não está malformado, mas é rejeitado pelo servidor do Stages. Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK chama `stage(didChange connectionState, withError error)` com código de erro = 1026 e IVS BroadcastErrorIsFatalKey = YES.

Ação: crie um token válido e tente entrar novamente.

Erros de rede para entrada inicial

Isso acontece quando o SDK não consegue entrar em contato com o servidor do Stages para estabelecer uma conexão. Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK chama `stage(didChange connectionState, withError error)` com código de erro = 1300 e IVS BroadcastErrorIsFatalKey = YES.

Ação: aguarde até que a conectividade do dispositivo se recupere e tente entrar novamente.

Erros de rede quando já ingressado

Se a conexão de rede do dispositivo cair, o SDK poderá perder sua conexão com os servidores de Estágios. Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK chama `stage(didChange connectionState, withError error)` com código de erro = 1300 e BroadcastErrorIsFatalKey valor IVS = SIM.

Ação: aguarde até que a conectividade do dispositivo se recupere e tente entrar novamente.

## Erros de publicação/assinatura

Inicial

Há vários erros:

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

Estes são relatados de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK repete a operação por um número limitado de vezes. as novas tentativas, o estado de publicação/assinatura é `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Se as tentativas de repetição forem bem-sucedidas, o estado mudará para `PUBLISHED` / `SUBSCRIBED`.

O SDK chama `IVSErrorSourceDelegate:didEmitError` com o código de erro relevante e IVS BroadcastErrorIsFatalKey = NO.

Ação: nenhuma ação é necessária, pois o SDK tenta novamente de forma automática. Opcionalmente, a aplicação pode atualizar a estratégia para forçar mais tentativas.

Já estabelecido, em seguida reprovar

Uma publicação ou assinatura pode falhar depois de ser estabelecida, provavelmente devido a um erro de rede. O código de erro para “conexão de peer perdida devido a um erro de rede” é 1400.

Isso é relatado de forma assíncrona por meio do renderizador de estágio fornecido pela aplicação.

O SDK tenta novamente a operação de publicação/assinatura. as novas tentativas, o estado de publicação/assinatura é `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Se as tentativas de repetição forem bem-sucedidas, o estado mudará para `PUBLISHED` / `SUBSCRIBED`.

O SDK chama `didEmitError` com código de erro = 1400 e IVS BroadcastErrorIsFatalKey = NO.

Ação: nenhuma ação é necessária, pois o SDK tenta novamente de forma automática. Opcionalmente, a aplicação pode atualizar a estratégia para forçar mais tentativas. Se houver perda total de conectividade, é provável que a conexão com o Stages também falhe.

# SDK de Transmissão do IVS: fontes de imagens personalizadas (streaming em tempo real)

As fontes de entrada de imagem personalizadas permitem que uma aplicação forneça a própria entrada de imagem para o SDK de transmissão, em vez de se limitar às câmeras definidas previamente. Uma fonte de imagem personalizada pode ser algo tão simples quanto uma marca d’água semitransparente ou uma cena estática de “volto logo”, ou pode permitir que a aplicação realize um processamento personalizado adicional, como a adição de filtros de beleza à câmera.

Quando você usa uma fonte de entrada de imagem personalizada para o controle personalizado da câmera (p. ex., o uso de bibliotecas de filtro de beleza que exigem acesso à câmera), o SDK de transmissão deixa de ser o responsável pelo gerenciamento da câmera. Em vez disso, a aplicação fica responsável por processar corretamente o ciclo de vida da câmera. Consulte a documentação oficial da plataforma sobre como sua aplicação deve gerenciar a câmera.

## Android

Após criar uma sessão `DeviceDiscovery`, crie uma fonte de entrada de imagem:

```
CustomImageSource imageSource = deviceDiscovery.createImageInputSource(new
 BroadcastConfiguration.Vec2(1280, 720));
```

Esse método retorna uma `CustomImageSource`, que é uma fonte de imagem corroborada por uma classe padrão Surface do Android. A subclasse `SurfaceSource` pode ser redimensionada e girada. Você também pode criar uma `ImagePreviewView` para exibir uma prévia de seu conteúdo.

Para recuperar a `Surface` subjacente:

```
Surface surface = surfaceSource.getInputSurface();
```

Essa `Surface` pode ser usada como o buffer de saída para produtores de imagens como Camera2, OpenGL ES e outras bibliotecas. O caso de uso mais simples é desenhar diretamente um bitmap estático ou uma cor na tela da Surface. No entanto, muitas bibliotecas (como bibliotecas de filtro de beleza) fornecem um método que permite que uma aplicação especifique uma `Surface` externa para renderização. Você pode usar tal método a fim de passar essa `Surface` para a biblioteca de filtros, permitindo que a biblioteca produza quadros processados para a sessão de transmissão transmitir.

Esta `CustomImageSource` pode ser agrupada em uma `LocalStageStream` e retornada pela `StageStrategy` para publicar em um `Stage`.

## iOS

Após criar uma sessão `DeviceDiscovery`, crie uma fonte de entrada de imagem:

```
let customSource = broadcastSession.createImageSource(withName: "customSourceName")
```

Esse método retorna uma `IVSCustomImageSource`, que é uma fonte de imagem que permite que a aplicação envie `CMSampleBuffers`manualmente. Para obter os formatos de pixel compatíveis, consulte a Referência do iOS Broadcast SDK; um link para a versão mais atual está disponível nas Notas de lançamento do Amazon IVS para a versão mais recente do Broadcast SDK.

As amostras enviadas para a fonte personalizada serão transmitidas para o Palco:

```
customSource.onSampleBuffer(sampleBuffer)
```

Para transmissão de vídeo, use esse método em um retorno de chamada. Por exemplo, se estiver usando a câmera, sempre que um novo buffer de amostra for recebido de uma `AVCaptureSession`, a aplicação pode encaminhar o buffer de amostra para a fonte de imagem

personalizada. Se desejar, a aplicação pode aplicar processamento adicional (como um filtro de beleza) antes de enviar a amostra para a fonte de imagem personalizada.

A `IVSCustomImageSource` pode ser agrupada em uma `IVSLocalStageStream` e retornada pela `IVSStageStrategy` para publicar em um `Stage`.

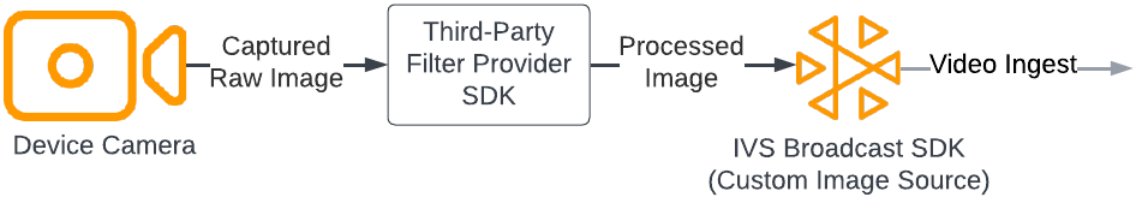
# SDK de Transmissão do IVS: filtros de câmera de terceiros (streaming em tempo real)

Este guia pressupõe que você já esteja familiarizado com fontes de imagem personalizada, bem como com a integração do SDK de Transmissão de streaming em tempo real do IVS com sua aplicação.

Os filtros de câmera permitem que os criadores de transmissões ao vivo aumentem ou alterem sua aparência facial ou de plano de fundo. Isso pode aumentar potencialmente o engajamento do espectador, atrair espectadores e aprimorar a experiência de transmissão ao vivo.

## Integração de filtros de câmera de terceiros

Você pode integrar SDKs de filtro de câmera de terceiros ao SDK de Transmissão do IVS alimentando a saída do SDK do filtro para uma fonte de entrada de imagem personalizada. Uma fonte de entrada de imagem personalizada permite que uma aplicação forneça a própria entrada de imagem para o SDK de Transmissão. O SDK de um provedor de filtro terceirizado pode gerenciar o ciclo de vida da câmera para processar imagens da câmera, aplicar um efeito de filtro e exibi-las em um formato que possa ser transmitido para uma fonte de imagem personalizada.




Consulte a documentação do seu fornecedor de filtro terceirizado para conhecer os métodos integrados de conversão de um quadro de câmera, com o efeito de filtro, aplicada a um formato que possa ser transmitido para uma fonte de entrada de imagem personalizada. O processo varia de acordo com a versão do SDK de Transmissão do IVS em uso:

- Web: o provedor do filtro deve ser capaz de renderizar sua saída em um elemento de tela. Assim, será possível usar o método captureStream para retornar um MediaStream do conteúdo da tela.

Em seguida, o MediaStream poderá ser convertido em uma instância de um LocalStageStream e publicado em um palco.

- Android: o SDK do provedor de filtro pode renderizar um quadro em um Android `Surface` fornecido pelo SDK do Transmissor do IVS ou converter o quadro em um bitmap. Se estiver usando um bitmap, ele poderá ser renderizado no `Surface` subjacente fornecido pela fonte de imagem personalizada ao desbloquear e gravar em uma tela.
- iOS: o SDK de um provedor de filtro terceirizado deve fornecer um quadro de câmera com um efeito de filtro aplicado como um `CMSampleBuffer`. Consulte a documentação do SDK do seu fornecedor de filtro terceirizado para obter informações sobre como fazer a `CMSampleBuffer` ser a saída final após o processamento da imagem da câmera.

## BytePlus

### Android

Instalar e configurar o SDK do BytePlus Effects

Consulte o Guia de acesso do BytePlus para Android para obter detalhes sobre como instalar, inicializar e configurar o SDK do BytePlus Effects.

Configurar a fonte de imagem personalizada

Após inicializar o SDK, alimente os quadros de câmera processados com um efeito de filtro aplicado a uma fonte de entrada de imagem personalizada. Para fazer isso, crie uma instância de um objeto `DeviceDiscovery` e crie uma fonte de imagem personalizada. Observe que quando você usa uma fonte de entrada de imagem personalizada para o controle personalizado da câmera, o SDK de Transmissão deixa de ser o responsável pelo gerenciamento da câmera. Em vez disso, a aplicação fica responsável por processar corretamente o ciclo de vida da câmera.

Java

```
var deviceDiscovery = DeviceDiscovery(applicationContext)
var customSource = deviceDiscovery.createImageInputSource( BroadcastConfiguration.Vec2(
720F, 1280F
))
var surface: Surface = customSource.inputSurface
var filterStream = ImageLocalStageStream(customSource)
```

Converter saída em bitmap e alimentar em fonte de entrada de imagem personalizada

Para permitir que os quadros da câmera com um efeito de filtro aplicado do SDK do BytePlus Effect sejam encaminhados diretamente para o SDK de Transmissão do IVS, converta a saída de uma textura do SDK do BytePlus Effects em um bitmap. Quando uma imagem for processada, o método `onDrawFrame()` será invocado pelo SDK. O método `onDrawFrame()` é um método público da interface GLSurfaceView.Renderer do Android. No aplicativo de amostra para Android fornecido pelo BytePlus, esse método é chamado em cada quadro da câmera e gera uma textura. Ao mesmo tempo, você pode complementar o método `onDrawFrame()` com a lógica para converter essa textura em um bitmap e alimentá-la em uma fonte de entrada de imagem personalizada. Conforme apresentado no exemplo de código a seguir, use o método `transferTextureToBitmap` fornecido pelo SDK do BytePlus para fazer essa conversão. Esse método é fornecido pela biblioteca com.bytedance.labcv.core.util.ImageUtil do SDK do BytePlus Effects, conforme apresentado na amostra de código a seguir. Em seguida, você poderá renderizar para o Android `Surface` subjacente de um `CustomImageSource` gravando o bitmap resultante na tela do Surface. Muitas invocações sucessivas de `onDrawFrame()` resultarão em uma sequência de bitmaps e, quando combinadas, criarão uma transmissão 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

Consulte o Guia de integração com Android do DeepAR para obter detalhes sobre como integrar o SDK do DeepAR com o SDK de Transmissão do IVS para Android.

## iOS

Consulte o Guia de integração com iOS do DeepAR para obter detalhes sobre como integrar o SDK do DeepAR com o SDK de Transmissão do IVS para iOS.

# Snap

## Web

Esta seção pressupõe que você já esteja familiarizado com a publicação e a inscrição em vídeos usando o SDK de Transmissão na Web.

Para integrar o SDK Camera Kit do Snap com o SDK de Transmissão na Web de streaming em tempo real do IVS, você precisa:

1. Instalar o SDK Camera Kit e o Webpack. (Nosso exemplo usa o Webpack como empacotador, mas você pode usar qualquer empacotador de sua escolha.)
2. Criar `index.html`.
3. Adicionar elementos de configuração.
4. Exibir e configurar os participantes.
5. Exibir câmeras e microfones conectados.
6. Criar uma sessão do Camera Kit.
7. Buscar e aplicar uma lente.
8. Renderizar a saída de uma sessão do Camera Kit em uma tela.
9. Fornecer uma fonte de mídia para o Camera Kit renderizar e publicar um `LocalStageStream`.
10. Criar um arquivo de configuração do Webpack.

Cada uma dessas etapas está descrita abaixo.

Instalar o SDK Camera Kit e o Webpack

```
npm i @snap/camera-kit webpack webpack-cli
```

Crie index.html

Em seguida, crie o padrão em HTML e importe o SDK de Transmissão da Web como uma tag de script. No código a seguir, certifique-se de substituir `<SDK version>` pela versão do SDK de Transmissão que você estiver usando.

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>
```

### Adicionar elementos de configuração

Crie o HTML para selecionar uma câmera e um microfone e especificar um 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>
```

Acrescente HTML adicional abaixo para exibir os feeds da câmera de participantes locais e 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>
```

Carregue lógica adicional, incluindo métodos auxiliares para configurar a câmera e o arquivo JavaScript incluído. (Posteriormente nesta seção, você criará esses arquivos JavaScript e os agrupará em um único arquivo, para poder importar o Camera Kit como um módulo. O arquivo

JavaScript incluído conterá a lógica para configurar o Camera Kit, aplicar uma lente e publicar o feed da câmera com uma lente aplicada a um palco.)

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>
```

Exibir e configurar os participantes

Em seguida, crie `helpers.js`, que contém métodos auxiliares que você usará para exibir e configurar os 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;
}
```

Exibir câmeras e microfones conectados

Em seguida, crie `media-devices.js`, que contém métodos auxiliares para exibir câmeras e microfones conectados ao seu 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,
    },
  });
}
```

Criar uma sessão do Camera Kit

Crie `stages.js`, que contém a lógica para aplicar uma lente ao feed da câmera e publicar o feed em um palco. Na primeira parte desse arquivo, importamos o SDK de Transmissão e o SDK do Camera Kit Web e inicializamos as variáveis que usaremos com cada SDK. Criaremos uma sessão do Camera Kit chamando `createSession` após fazer o bootstrap do SDK do Camera Kit Web. Observe que um objeto de elemento de tela é transmitido para uma sessão e isso faz com que o Camera Kit seja renderizado nessa tela.

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 });
```

### Buscar e aplicar uma lente

Para buscar suas lentes, insira seu ID de grupo de lentes, que está disponível no Portal do Desenvolvedor do Camera Kit. Nesse exemplo, simplificamos as coisas aplicando a primeira lente na matriz de lentes que é retornada.

JavaScript

```
const { lenses } = await cameraKit.lensRepository.loadLensGroups([
    INSERT_LENS_GROUP_ID_HERE,
  ]);

  session.applyLens(lenses[0]);
```

Renderizar a saída de uma sessão do Camera Kit em uma tela

Use o método captureStream para retornar um `MediaStream` do conteúdo da tela. A tela conterá uma transmissão de vídeo do feed da câmera com uma lente aplicada. Além disso, adicione receptores de eventos para os botões de silenciar a câmera e o microfone, bem como receptores de eventos para entrar e sair de um palco. Caso um receptor entre em um palco, passaremos uma sessão do Camera Kit e o `MediaStream` da tela para que ela possa ser publicada em um palco.

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();
  });
};
```

Fornecer o Camera com uma fonte de mídia para renderização e publicar um LocalStageStream

Para publicar uma transmissão de vídeo com uma lente aplicada, crie uma função chamada `setCameraKitSource` para transmitir o `MediaStream` que foi capturado da tela anteriormente. O `MediaStream` da tela não estará fazendo nada no momento porque ainda não incorporamos nosso feed de câmera local. Podemos incorporar nosso feed de câmera local chamando o método auxiliar `getCamera` e atribuindo-o a `localCamera`. Em seguida, podemos transmitir nosso feed de câmera local (via `localCamera`) e o objeto da sessão para `setCameraKitSource`.

A função `setCameraKitSource` converte nosso feed de câmera local em uma fonte de mídia para o CameraKit chamando `createMediaStreamSource`. Em seguida, a fonte de mídia para `CameraKit` é transformada para espelhar a câmera frontal. O efeito da lente é aplicado à fonte de mídia e renderizado na tela de saída chamando `session.play()`.

Agora, com a lente aplicada ao `MediaStream` capturado da tela, poderemos publicá-lo em um palco. Fazemos isso criando um `LocalStageStream` com as faixas de vídeo do `MediaStream`. Em seguida, é possível transmitir uma instância de `LocalStageStream` para um `StageStrategy` para publicação.

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;
    },
  };
```

O código restante abaixo serve para criar e gerenciar nosso palco:

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();
```

Criar um arquivo de configuração do Webpack

Crie `webpack.config.js` e adicione o código a seguir. Isso empacota a lógica acima para que você possa usar a instrução de importar para usar o Camera Kit.

JavaScript

```
const path = require('path');
module.exports = {
  entry: ['./stage.js'],
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

Por fim, execute `npm run build` para empacotar seu JavaScript conforme definido no arquivo de configuração do Webpack. Em seguida, você poderá fornecer em HTML e JavaScript de um servidor Web. Por exemplo, você poderá usar o servidor HTTP do Python e abrir localhost:8000 para ver o resultado:

```
# Run this from the command line and the directory containing index.html
python3 -m http.server -d ./
```

## Android

Para integrar o SDK do Camera Kit do Snap com o SDK de Transmissão do IVS para Android, você deverá instalar o SDK do Camera Kit, inicializar uma sessão do Camera Kit, aplicar uma lente e alimentar a saída da sessão do Camera Kit para a fonte de entrada de imagem personalizada.

Para instalar o SDK do Camera Kit, adicione o seguinte ao arquivo `build.gradle` do seu módulo. Substitua `$cameraKitVersion` pela versão mais recente do SDK do Camera Kit.

Java

```
implementation "com.snap.camerakit:camerakit:$cameraKitVersion"
```

Inicialize e obtenha um `cameraKitSession`. O Camera Kit também fornece um pacote conveniente para as APIs CameraX do Android, de modo que você não precise escrever uma lógica complicada para usar o CameraX com o Camera Kit. Você pode usar o objeto `CameraXImageProcessorSource` como uma fonte para ImageProcessor, o que permite iniciar quadros de streaming de pré-visualização da câmera.

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();
    }
```

Buscar e aplicar lentes

Você pode configurar as lentes e sua ordem no carrossel no Portal do Desenvolvedor do Camera Kit:

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, envie quadros processados para o `Surface` subjacente de uma fonte de imagem personalizada. Use um objeto `DeviceDiscovery` e crie um `CustomImageSource` para retornar um `SurfaceSource`. Em seguida, você poderá renderizar a saída de uma sessão `CameraKit` para o `Surface` subjacente fornecido pelo `SurfaceSource`.

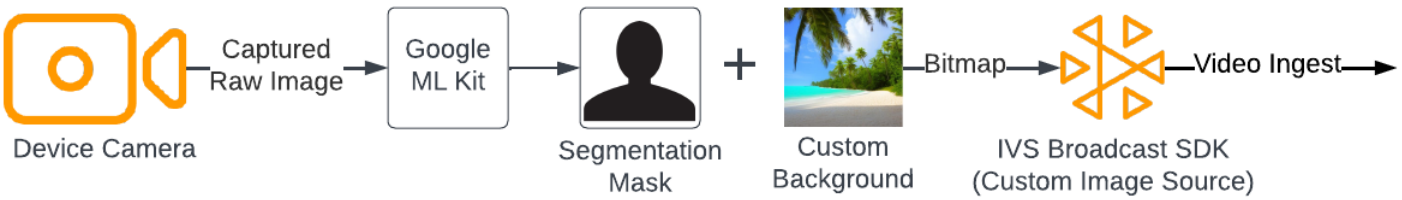
Java

```
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
```

## Substituição de plano de fundo

A substituição do fundo é um tipo de filtro de câmera que permite que criadores de transmissões ao vivo alterem seus planos de plano de fundo. Conforme exibido no diagrama a seguir, a substituição do plano de fundo envolve:

1. Obter uma imagem da câmera com base no feed da câmera ao vivo.
2. Segmentá-la em componentes de primeiro e segundo plano usando o Google ML Kit.
3. Combinar a máscara de segmentação resultante com uma imagem de plano de fundo personalizada.
4. Transmiti-la para uma fonte de imagem personalizada para transmissão.



## Web

Esta seção pressupõe que você já esteja familiarizado com a publicação e a inscrição em vídeos usando o SDK de Transmissão na Web.

Para substituir o plano de fundo de uma transmissão ao vivo por uma imagem personalizada, use o modelo de segmentação de selfies com o MediaPipe Image Segmenter. Esse é um modelo de machine-learning que identifica quais pixels no quadro do vídeo estão em primeiro ou segundo plano. Em seguida, você pode usar os resultados do modelo para substituir o fundo de uma transmissão ao vivo, copiando os pixels do primeiro plano do feed de vídeo para uma imagem personalizada representando o novo plano de fundo.

Para integrar a substituição em segundo plano com o SDK de Transmissão na Web de streaming em tempo real do IVS, você precisará:

1. Instalar o MediaPipe e o Webpack. (Nosso exemplo usa o Webpack como empacotador, mas você pode usar qualquer empacotador de sua escolha.)
2. Criar `index.html`.
3. Adiciona elementos de mídia.
4. Adicionar uma tag de script.
5. Criar `app.js`.
6. Carregar uma imagem de plano de fundo personalizada.
7. Crie uma instância de `ImageSegmenter`.
8. Renderizar o feed de vídeo em uma tela.
9. Criar uma lógica de substituição em segundo plano.
10. Criar um arquivo de configuração do Webpack.
11. Empacotar seu arquivo JavaScript.

### Instalar o MediaPipe e o Webpack

Para começar, instale os pacotes npm `@mediapipe/tasks-vision` e `webpack`. O exemplo abaixo usa o Webpack como um empacotador de JavaScript, mas você pode usar um empacotador diferente se quiser.

JavaScript

```
npm i @mediapipe/tasks-vision webpack webpack-cli
```

Certifique-se também de atualizar seu `package.json` para especificar `webpack` como seu script de compilação:

JavaScript

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
```

Crie index.html

Em seguida, crie o padrão em HTML e importe o SDK de Transmissão da Web como uma tag de script. No código a seguir, certifique-se de substituir `<SDK version>` pela versão do SDK de Transmissão que você estiver usando.

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>
```

### Adicionar elementos de mídia

Em seguida, adicione um elemento de vídeo e dois elementos de tela na tag do corpo. O elemento de vídeo conterá o feed da câmera ao vivo e será usado como entrada para o MediaPipe Image Segmenter. O primeiro elemento de tela será usado para renderizar uma prévia do feed que será transmitido. O segundo elemento de tela será usado para renderizar a imagem personalizada que será usada como plano de fundo. Como a segunda tela com a imagem personalizada é usada somente como fonte para copiar programaticamente pixels dela para a tela final, ela ficará oculta.

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>
```

### Adicionar uma tag de script

Adicione uma tag de script para carregar um arquivo JavaScript incluído que conterá o código para fazer a substituição em segundo plano e publicá-lo em um palco:

```
<script src="./dist/bundle.js"></script>
```

### Criar app.js

Em seguida, crie um arquivo JavaScript para obter os objetos dos elementos da tela e do vídeo que foram criados na página HTML. Importe os módulos `ImageSegmenter` e `FilesetResolver`. O módulo `ImageSegmenter` será usado para realizar a tarefa de segmentação.

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";
```

Em seguida, crie uma função chamada `init()` para recuperar o MediaStream da câmera do usuário e invoque uma função de retorno de chamada sempre que o quadro da câmera terminar o carregamento. Adicione receptores de eventos para os botões de entrar e sair de um palco.

Observe que, ao entrar em um palco, transmitimos uma variável chamada `segmentationStream`. Trata-se de uma transmissão de vídeo capturado de um elemento de tela, contendo uma imagem de primeiro plano sobreposta à imagem personalizada que representa o plano de fundo. Posteriormente, essa transmissão personalizada será usada para criar uma instância de um `LocalStageStream`, que poderá ser publicada em um palco.

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();
  });
};
```

Carregar uma imagem de plano de fundo personalizada

Na parte inferior da função `init`, adicione código para chamar uma função `initBackgroundCanvas`, que carrega uma imagem personalizada de um arquivo local e a renderiza em uma tela. Definiremos essa função na próxima etapa. Atribua o `MediaStream` recuperado da câmera do usuário ao objeto de vídeo. Posteriormente, esse objeto de vídeo será passado para o Image Segmenter. Além disso, defina uma função chamada `renderVideoToCanvas` como a função de retorno de chamada a ser invocada sempre que um quadro de vídeo terminar o carregamento. Definiremos essa função em uma etapa posterior.

JavaScript

```
initBackgroundCanvas();

  video.srcObject = localCamera;
  video.addEventListener("loadeddata", renderVideoToCanvas);
```

Vamos implementar a função `initBackgroundCanvas`, que carrega uma imagem de um arquivo local. Neste exemplo, usamos a imagem de uma praia como plano de fundo personalizado. A tela contendo a imagem personalizada ficará oculta da exibição, pois você a mesclará com os pixels do primeiro plano do elemento de tela que contém o feed da câmera.

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);
  };
};
```

### Criar uma instância do ImageSegmenter

Em seguida, crie uma instância de `ImageSegmenter`, que segmentará a imagem e retornará o resultado como uma máscara. Ao criar uma instância de um `ImageSegmenter`, você usará o modelo de segmentação 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,
  });
};
```

### Renderizar o feed de vídeo em uma tela

Em seguida, crie a função que renderiza o feed de vídeo para o outro elemento da tela. Precisamos renderizar o feed de vídeo em uma tela para que possamos extrair os pixels do primeiro plano usando a API Canvas 2D. Ao fazer isso, também transmitiremos um quadro de vídeo para nossa instância de `ImageSegmenter`, usando o método segmentforVideo para segmentar o primeiro plano em relação ao de fundo no quadro do vídeo. Quando o método segmentforVideo retornar, ele invocará nossa função personalizada de retorno de chamada, `replaceBackground`, para fazer a substituição em segundo plano.

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);
};
```

Criar uma lógica de substituição em segundo plano

Crie a função `replaceBackground`, que mescla a imagem de plano de fundo personalizada com o primeiro plano do feed da câmera para substituir o plano de fundo. Primeiro, a função recupera os dados de pixel subjacentes da imagem de plano de fundo personalizada e do feed de vídeo dos dois elementos de tela criados anteriormente. Em seguida, ela fará uma iteração pela máscara fornecida por `ImageSegmenter`, que indica quais pixels estão em primeiro plano. Conforme percorre a máscara, ela copiará seletivamente os pixels que contenham o feed da câmera do usuário para os dados de pixels de plano de fundo correspondentes. Depois disso, ela converterá os dados finais de pixel com o primeiro plano copiado para o plano de fundo e os desenhará em uma tela.

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);
}
```

Para referência, aqui está o arquivo `app.js` completo contendo toda a lógica acima:

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();
```

### Criar um arquivo de configuração do Webpack

Adicione essa configuração ao arquivo de configuração do Webpack para empacotar `app.js`, de modo que as chamadas de importação funcionem:

JavaScript

```
const path = require("path");
module.exports = {
  entry: ["./app.js"],
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist"),
  },
};
```

Empacotar seus arquivos JavaScript

```
npm run build
```

Inicie um servidor HTTP simples no diretório que contém `index.html` e abra `localhost:8000` para ver o resultado:

```
python3 -m http.server -d ./
```

## Android

Para substituir o plano de fundo em sua transmissão ao vivo, você pode usar a API de segmentação de selfies do Google ML Kit. A API de segmentação de selfies aceita uma imagem da câmera como entrada e retorna uma máscara que fornece uma pontuação de confiança para cada pixel da imagem, indicando se ela estava em primeiro plano ou em segundo plano. Com base na pontuação de confiança, você poderá recuperar a cor de pixel correspondente da imagem de plano de fundo ou da imagem de primeiro plano. Esse processo continua até que todas as pontuações de confiança na máscara tenham sido examinadas. O resultado será uma nova matriz de cores de pixels contendo pixels em primeiro plano combinados com pixels da imagem de plano de fundo.

Para integrar a substituição em segundo plano com o SDK de Transmissão para Android de streaming em tempo real do IVS, você precisará:

1. Instalar as bibliotecas CameraX e o Google ML Kit.
2. Inicializar variáveis clichê.
3. Criar uma fonte de imagens personalizada.
4. Gerenciar os quadros da câmera.
5. Transmitir as molduras da câmera para o Google ML Kit.
6. Sobrepor o primeiro plano da moldura da câmera ao seu plano de fundo personalizado.
7. Alimentar a nova imagem para uma fonte de imagem personalizada.

### Instalar as bibliotecas CameraX e o Google ML Kit

Para extrair imagens do feed da câmera ao vivo, use a biblioteca CameraX do Android. Para instalar a biblioteca CameraX e o Gooogle ML Kit, adicione o seguinte ao arquivo `build.gradle` do seu módulo. Substitua `${camerax_version}` e `${google_ml_kit_version}` pela versão mais recente das bibliotecas CameraX e 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 as seguintes 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
```

Inicializar variáveis clichê

Inicialize uma instância de `ImageAnalysis` e uma instância de `ExecutorService`:

Java

```
private lateinit var binding: ActivityMainBinding
private lateinit var cameraExecutor: ExecutorService
private var analysisUseCase: ImageAnalysis? = null
```

Inicialize uma instância do Segmenter em STREAM_MODE:

Java

```
private val options =
        SelfieSegmenterOptions.Builder()
            .setDetectorMode(SelfieSegmenterOptions.STREAM_MODE)
            .build()

private val segmenter = Segmentation.getClient(options)
```

Criar uma fonte de imagens personalizada

No método on`Create` da sua atividade, crie uma instância de um objeto `DeviceDiscovery` e crie uma fonte de imagem personalizada. O `Surface` fornecido pela Fonte de imagem personalizada receberá a imagem final, com o primeiro plano sobreposto a uma imagem de plano de fundo

personalizada. Em seguida, você criará uma instância de um `ImageLocalStageStream` usando a fonte de imagem personalizada. Em seguida, a instância de um `ImageLocalStageStream` (nomeada `filterStream`, neste exemplo) poderá ser publicada em um palco. Consulte o Guia do SDK de Transmissão do IVS para Android para obter instruções sobre como configurar um palco. Por fim, crie também um tópico que será usado para gerenciar a câmera.

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()
```

Gerenciar os quadros da câmera

Em seguida, crie uma função para inicializar a câmera. Essa função usa a biblioteca CameraX para extrair imagens do feed da câmera ao vivo. Primeiro, você cria uma instância de um `ProcessCameraProvider` chamado `cameraProviderFuture`. Esse objeto representa um resultado futuro da obtenção de um fornecedor de câmeras. Em seguida, você carrega uma imagem do seu projeto como um bitmap. Este exemplo usa a imagem de uma praia como plano de fundo, mas pode ser qualquer imagem que você quiser.

Em seguida, você adiciona um receptor a `cameraProviderFuture`. Esse receptor será notificado quando a câmera ficar disponível ou se ocorrer um erro durante o processo de obtenção de um receptor de câmeras.

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))
    }
```

No receptor , crie `ImageAnalysis.Builder` para acessar cada quadro individual do feed da câmera ao vivo. Defina a estratégia de contrapressão como `STRATEGY_KEEP_ONLY_LATEST`. Isso

garante que apenas um quadro de câmera seja entregue para processamento por vez. Converta cada quadro individual da câmera em um bitmap, para que você possa extrair seus pixels e depois combiná-los com a imagem de plano de fundo 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())
```

Transmitir as molduras da câmera para o Google ML Kit

Em seguida, crie um `InputImage` e passe-o para a instância do Segmenter para processamento. Um `InputImage` pode ser criado com base em um `ImageProxy` fornecido pela instância de `ImageAnalysis`. Após o fornecimento de um `InputImage` ao Segmenter, ele retornará uma máscara com pontuações de confiança indicando a probabilidade de um pixel estar em primeiro plano ou em segundo plano. Essa máscara também fornece propriedades de largura e altura, que você usará para criar uma nova matriz contendo os pixels de plano de fundo da imagem de plano de fundo personalizada carregada 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)
```

Sobrepor o primeiro plano da moldura da câmera ao seu plano de fundo personalizado

Com a máscara contendo as pontuações de confiança, a moldura da câmera como um bitmap e os pixels coloridos da imagem de plano de fundo personalizada, você tem tudo o que precisa para sobrepor o primeiro plano ao plano de fundo personalizado. Em seguida, a função `overlayForeground` será chamada com os seguintes parâmetros:

Java

```
resultBitmap = overlayForeground(mask, maskWidth, maskHeight, inputBitmap,
 backgroundPixels)
```

Essa função percorre a máscara e verifica os valores de confiança para determinar se deseja obter a cor de pixel correspondente da imagem de plano de fundo ou da moldura da câmera. Se o valor de confiança indicar a probabilidade de que um pixel na máscara esteja em segundo plano, ele obterá a cor de pixel correspondente da imagem de plano de fundo; caso contrário, obterá a cor de pixel correspondente da moldura da câmera para criar o primeiro plano. Quando a função terminar a iteração pela máscara, um novo bitmap será criado usando a nova matriz de pixels coloridos e retornado. Esse novo bitmap vai conter o primeiro plano sobreposto ao plano de fundo 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
        )
    }
```

Alimentar a nova imagem para uma fonte de imagem personalizada

Em seguida, você poderá gravar o novo bitmap no `Surface` fornecido por uma fonte de imagem personalizada. Isso o transmitirá para o seu palco.

Java

```
resultBitmap = overlayForeground(mask, inputBitmap, mutableBitmap, bgBitmap)
canvas = surface.lockCanvas(null);
canvas.drawBitmap(resultBitmap, 0f, 0f, null)
```

Aqui está a função completa para obter os quadros da câmera, transmiti-los para o Segmenter e sobrepô-los ao plano de fundo:

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 Transmissão do IVS: modos de áudio móvel (streaming em tempo real)

A qualidade do áudio é uma parte importante de qualquer experiência de mídia em tempo real, e não há uma configuração de áudio única que funcione melhor para cada caso de uso. Para garantir que seus usuários tenham a melhor experiência ao ouvir uma transmissão em tempo real do IVS, nossos SDKs móveis oferecem várias configurações de áudio predefinidas, bem como personalizações mais poderosas, conforme necessário.

## Introdução

Os SDKs de transmissão móvel do IVS oferecem uma classe `StageAudioManager`. Essa classe foi projetada para ser o único ponto de contato para controlar os modos de áudio subjacentes em ambas as plataformas. No Android, isso controla o AudioManager, incluindo o modo de áudio, a fonte do áudio, o tipo de conteúdo, o uso e os dispositivos de comunicação. No iOS, o elemento controla o a aplicação AVAudioSession, bem como se o voiceProcessing está habilitado.

Importante: não interaja com `AVAudioSession` ou `AudioManager` diretamente enquanto o SDK de Transmissão em tempo real do IVS estiver ativo. Isso poderá resultar na perda de áudio ou na gravação ou reprodução do áudio no dispositivo errado.

Antes de criar seu primeiro objeto `DeviceDiscovery` ou `Stage`, é necessário configurar classe `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
```

Se nada for definido em `StageAudioManager` antes da inicialização de uma instância `DeviceDiscovery` ou `Stage`, a predefinição `VideoChat` será aplicada automaticamente.

## Predefinições do modo de áudio

O SDK de Transmissão em tempo real fornece três predefinições, cada uma personalizada para casos de uso comuns, conforme descrito abaixo. Para cada predefinição, abordamos cinco categorias principais que diferenciam as predefinições umas das outras.

### Bate-papo por vídeo

Essa é a predefinição padrão, projetada para quando o dispositivo local tiver uma conversa em tempo real com outros participantes.

| Categoria | Android | iOS |
| --- | --- | --- |
| Cancelamento de eco | Habilitado | Habilitado |
| Alternador de volume | Volume da chamada | Volume da chamada |

| Categoria | Android | iOS |
| --- | --- | --- |
| Seleção de microfone | Limitado com base no sistema operacional. Os microfones USB podem não estar disponíveis. | Limitado com base no sistema operacional. Os microfones USB e Bluetooth podem não estar disponíveis.<br><br>Os fones de ouvido Bluetooth que processam entrada e saída juntas devem funcionar (por exemplo, AirPods). |
| Saída de áudio | Qualquer dispositivo de saída deve funcionar. | Limitado com base no sistema operacional. Fones de ouvido com fio podem não estar disponíveis. |
| Qualidade de áudio | Média/baixa. Soará como um telefonema, não como uma reprodução de mídia. | Média/baixa. Soará como um telefonema, não como uma reprodução de mídia. |

## Somente inscrição

Essa predefinição foi criada para quando você planeja se inscrever em outros participantes da publicação, mas não publicar a si mesmo. Ela se concentra na qualidade do áudio e na compatibilidade com todos os dispositivos de saída disponíveis.

| Categoria | Android | iOS |
| --- | --- | --- |
| Cancelamento de eco | Desabilitado | Desabilitado |
| Alternador de volume | Volume de mídia | Volume de mídia |
| Seleção de microfone | N/D, essa predefinição não foi projetada para publicação. | N/D, essa predefinição não foi projetada para publicação. |
| Saída de áudio | Qualquer dispositivo de saída deve funcionar. | Qualquer dispositivo de saída deve funcionar. |

| Categoria | Android | iOS |
| --- | --- | --- |
| Qualidade de áudio | Alta. Qualquer tipo de mídia deve ser transmitido com clareza, inclusive música. | Alta. Qualquer tipo de mídia deve ser transmitido com clareza, inclusive música. |

## Studio

Essa predefinição foi projetada para inscrições de alta qualidade, mantendo a capacidade de publicação. É necessário que o hardware de gravação e reprodução forneça o cancelamento de eco. Um caso de uso aqui seria usar um microfone USB e um fone de ouvido com fio. O SDK manterá a mais alta qualidade de áudio e, ao mesmo tempo, dependerá da separação física desses dispositivos contra a ocorrência de eco.

| Categoria | Android | iOS |
| --- | --- | --- |
| Cancelamento de eco | Desabilitado | Desabilitado |
| Alternador de volume | Volume de mídia na maioria dos casos. Volume da chamada quando um microfone Bluetooth estiver conectado. | Volume de mídia |
| Seleção de microfone | Qualquer microfone deve funcionar. | Qualquer microfone deve funcionar. |
| Saída de áudio | Qualquer dispositivo de saída deve funcionar. | Qualquer dispositivo de saída deve funcionar. |
| Qualidade de áudio | Alta. Ambos os lados devem ser capazes de enviar música e ouvi-la claramente do outro lado.<br><br>Quando um fone de ouvido Bluetooth for conectado, a qualidade do áudio diminuirá devido à ativação do modo SCO do Bluetooth. | Alta. Ambos os lados devem ser capazes de enviar música e ouvi-la claramente do outro lado.<br><br>Quando um fone de ouvido Bluetooth for conectado, a qualidade do áudio poderá diminuir devido à ativação |

| Categoria | Android | iOS |
| --- | --- | --- |
| | | do modo SCO do Bluetooth, dependendo do fone de ouvido. |

## Casos de uso avançados

Além das predefinições, os SDKs de Transmissão de streaming em tempo real para iOS e Android permitem configurar os modos de áudio da plataforma subjacente:

- No Android, defina AudioSource, Usage e ContentType.
- No iOS, use AVAudioSession.Category, AVAudioSession.CategoryOptions, AVAudioSession.Mode e a capacidade de alternar se o processamento de voz está habilitado ou não durante a publicação.

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
```

### Publicação com Bluetooth no Android

O SDK reverterá automaticamente para a predefinição `VIDEO_CHAT` no Android quando as condições a seguir forem atendidas:

- A configuração atribuída não usar o valor de uso `VOICE_COMMUNICATION`.
- Houver um microfone Bluetooth conectado ao dispositivo.
- O participante local estiver publicando em um palco.

Essa é uma limitação do sistema operacional Android em relação à forma como os fones de ouvido Bluetooth são usados para gravar áudio.

## Integração com outros SDKs

Como o iOS e o Android são compatíveis apenas com um modo de áudio ativo por aplicação, é comum entrar em conflito se o aplicativo usar vários SDKs que exijam controle do modo de áudio. Veja abaixo algumas estratégias comuns de resolução para tentar solucionar esses conflitos.

### Combinar os valores do modo de áudio

Usando as opções avançadas de configuração de áudio do SDK do IVS ou a funcionalidade de outro SDK, faça com que os dois SDKs se alinhem aos valores subjacentes.

### Agora

iOS

No iOS, pedir que o SDK do Agora que mantenha o `AVAudioSession` ativo impedirá que ele seja desativado enquanto o SDK de Transmissão de streaming em tempo real do IVS estiver fazendo uso dele.

```
myRtcEngine.SetParameters("{\"che.audio.keep.audiosession\":true}");
```

### Android

Evite chamar `setEnableSpeakerphone` em `RtcEngine` e chamar `enableLocalAudio(false)` enquanto publica com o SDK de Transmissão de streaming em tempo real do IVS. Você pode chamar `enableLocalAudio(true)` novamente quando o SDK do IVS não estiver sendo publicado.

# Uso do Amazon EventBridge com o streaming em tempo real do IVS

Você pode usar o Amazon EventBridge para monitorar seus streams do Amazon Interactive Video Service (IVS).

O Amazon IVS envia eventos de alteração sobre o status de seus streams para o Amazon EventBridge. Todos os eventos que são fornecidos são válidos. No entanto, os eventos são enviados em uma base de melhor esforço, o que significa que não há garantia de que:

- Os eventos serão entregues: um evento designado pode ocorrer (por exemplo, um participante realiza uma publicação), mas é possível que o Amazon IVS não envie um evento correspondente para o EventBridge. O Amazon IVS tenta entregar eventos por várias horas antes de desistir.
- Os eventos que são entregues vão chegar em um período especificado: você pode receber eventos com até algumas horas de atraso.
- Os eventos serão entregues em ordem: os eventos podem estar fora de ordem, especialmente se forem enviados com um curto intervalo de tempo. Por exemplo, você poderá ver participante não publicado antes de visualizá-lo como publicado.

Embora seja raro que os eventos estejam ausentes, atrasados ou fora de sequência, você deve lidar com essas possibilidades se você escrever programas críticos para os negócios que dependem da ordem ou da existência de eventos de notificação.

Você pode criar regras do EventBridge para qualquer um dos seguintes eventos.

| Tipo de evento | Evento | Enviado quando … |
| --- | --- | --- |
| Alteração do estado de composição do IVS | Falha no destino | Uma tentativa de envio para um destino falhou. Por exemplo, a transmissão para um canal falhou porque não havia chave de transmissão ou porque outra transmissão estava acontecendo. |
| Alteração do estado de composição do IVS | Início do destino | A saída para um destino foi iniciada com sucesso. |

| Tipo de evento | Evento | Enviado quando … |
| --- | --- | --- |
| Alteração do estado de composição do IVS | Fim do destino | Conclusão da saída para um destino. |
| Alteração do estado de composição do IVS | Reconexão de destino | A saída para um destino foi interrompida e há uma tentativa de reconexão em andamento. |
| Alteração do estado de composição do IVS | Início da sessão | Uma sessão de composição foi criada. Esse evento é acionado quando um pipeline de processo de composição é inicializado com sucesso. Nesse momento, o funil de composição terá feito a inscrição com sucesso em um palco, estará recebendo mídia e será capaz de compor vídeos. |
| Alteração do estado de composição do IVS | Término da sessão | Uma sessão de composição concluída. |
| Alteração do estado de composição do IVS | Falha na sessão | Falha na inicialização de um pipeline de composição devido à indisponibilidade dos recursos do palco ou a qualquer outro erro interno. |
| Atualização de palco do IVS | Publicação por participante | Um participante começa a publicar em um palco. |
| Atualização de palco do IVS | Publicação interrompida por participante | Um participante parou de publicar em um palco. |

# Criação de regras do Amazon EventBridge para o Amazon IVS

Você pode criar uma regra do que é acionado em um evento emitido pelo Amazon IVS. Siga as etapas em Create a rule in Amazon EventBridge no Guia do usuário do Amazon EventBridge. Ao selecionar um serviço, escolha Interactive Video Service (IVS).

# Exemplos: alteração do estado de composição do IVS

Falha no destino: esse evento é enviado quando uma tentativa de saída para um destino falha. Por exemplo, a transmissão para um canal falhou porque não havia chave de transmissão ou porque outra transmissão estava acontecendo.

```
{
   "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"
   }
}
```

Início do destino: esse evento é enviado quando a saída para um destino é iniciada com sucesso.

```
{
   "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>",
   }
}
```

Fim do destino: esse evento é enviado quando a saída para um destino é concluída.

```
{
   "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>",
   }
}
```

Reconexão de destino: esse evento é enviado quando a saída para um destino é interrompida e há uma tentativa de reconexão.

```
{
   "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>",
    }
}
```

Início da sessão: esse evento é enviado quando uma sessão de composição foi criada. Esse evento é acionado quando um pipeline de processo de composição é inicializado com sucesso. Nesse momento, o funil de composição terá feito a inscrição com sucesso em um palco, estará recebendo mídia e será capaz de compor vídeos.

```
{
   "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>"
   }
}
```

Fim da sessão: esse evento é enviado quando uma sessão de composição é concluída e todos os recursos são excluídos.

```
{
   "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>"
    }
}
```

Falha na sessão: esse evento é enviado quando um pipeline de composição falha na inicialização devido à falta de recursos do palco, à falta de participantes no palco ou a qualquer outro erro 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"
    }
}
```

# Exemplos: atualização de palco

Os eventos de atualização de palco incluem um nome de evento (que classifica o evento) e metadados sobre o evento. Os metadados incluem o ID do participante que acionou o evento, os IDs de sessão e palco associados e o ID do usuário.

Publicação por participante: este evento é enviado quando um participante começa a publicar em um palco.

```
{
    "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"
    }
}
```

Publicação interrompida por participante: este evento é enviado quando um participante para de publicar em um palco.

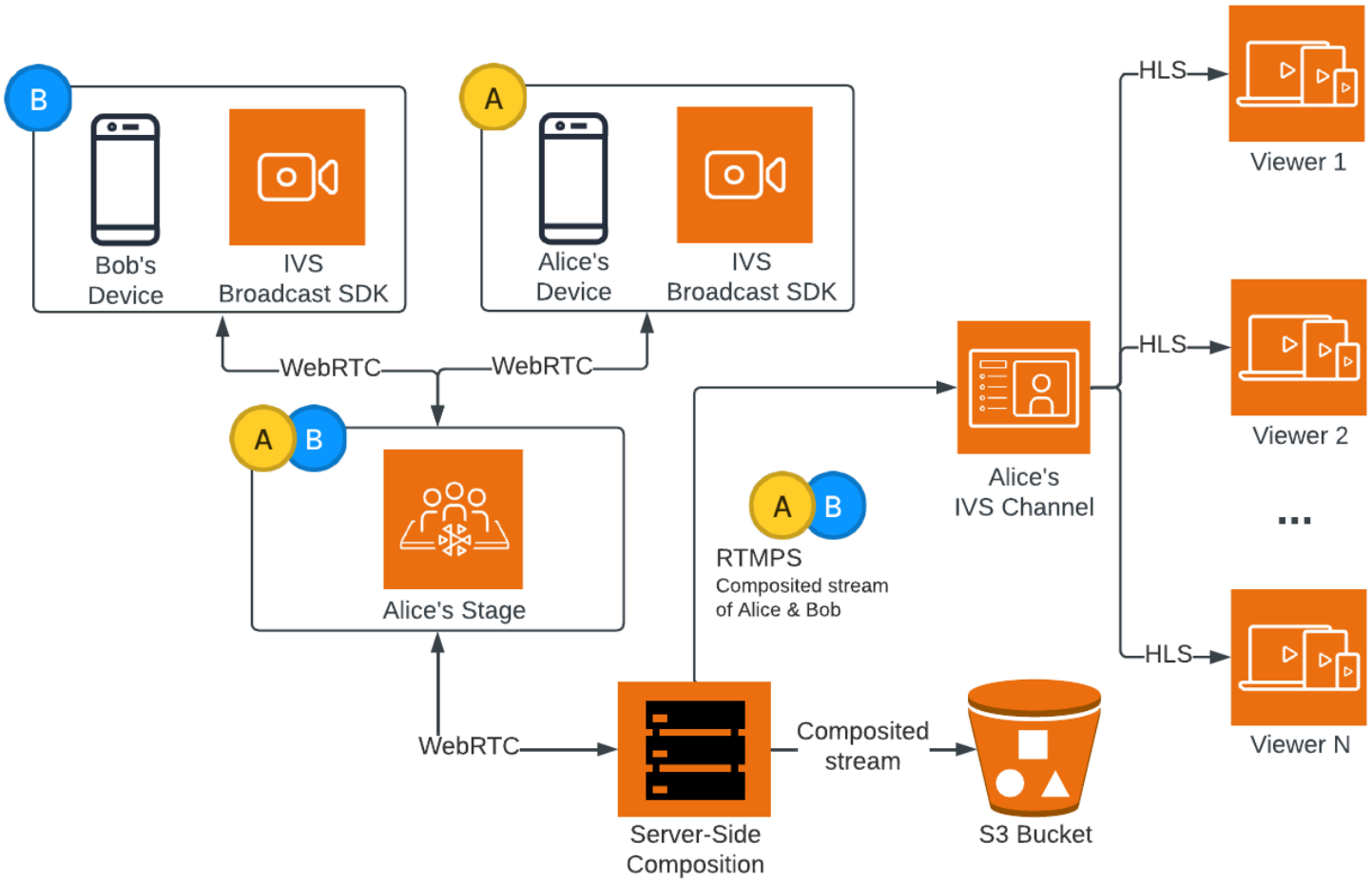
```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-1234567890",
        "event_name": "Participant Unpublished",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f"
    }
}
```

# Composição do servidor (streaming em tempo real)

A composição do servidor usa um servidor do IVS para mixar áudio e vídeo de todos os participantes do palco e, em seguida, enviar esse vídeo mixado para um canal do IVS (por exemplo, para alcançar um público maior) ou para um bucket S3.. A composição do servidor é invocada por meio de endpoints do ambiente de gerenciamento do IVS na região de origem do palco.

A transmissão ou gravação de um palco usando a composição do servidor oferece vários benefícios, fazendo dela uma opção atraente para usuários que buscam fluxos de trabalho de vídeo eficientes e confiáveis na nuvem.

Este diagrama ilustra o funcionamento da composição do servidor:




# Benefícios

Comparada à composição do lado do cliente, a composição do servidor tem os seguintes benefícios:

- Redução da carga do cliente: com a composição do servidor, a carga do processamento e da combinação de fontes de áudio e vídeo é transferida dos dispositivos individuais do cliente para o próprio servidor. A composição do servidor elimina a necessidade de dispositivos clientes usarem seus recursos de CPU e rede para compor a visualização e transmiti-la ao IVS. Isso significa que os espectadores podem assistir à transmissão sem que seus dispositivos precisem processar tarefas que consomem muitos recursos, o que pode levar a uma maior duração da bateria e a experiências de visualização mais uniformes.
- Qualidade consistente: a composição do servidor permite um controle preciso sobre a qualidade, a resolução e a taxa de bits do fluxo final. Isso garante uma experiência de visualização consistente para todos os espectadores, independentemente das capacidades individuais de seus dispositivos.
- Resiliência: ao centralizar o processo de composição no servidor, a transmissão se torna mais robusta. Mesmo que o dispositivo publicador tenha limitações ou flutuações técnicas, o servidor poderá se adaptar e fornecer uma transmissão mais suave para todo o público.
- Eficiência de largura de banda: como o servidor processa a composição, os publicadores de palco não precisam gastar mais largura de banda transmitindo o vídeo para o IVS.

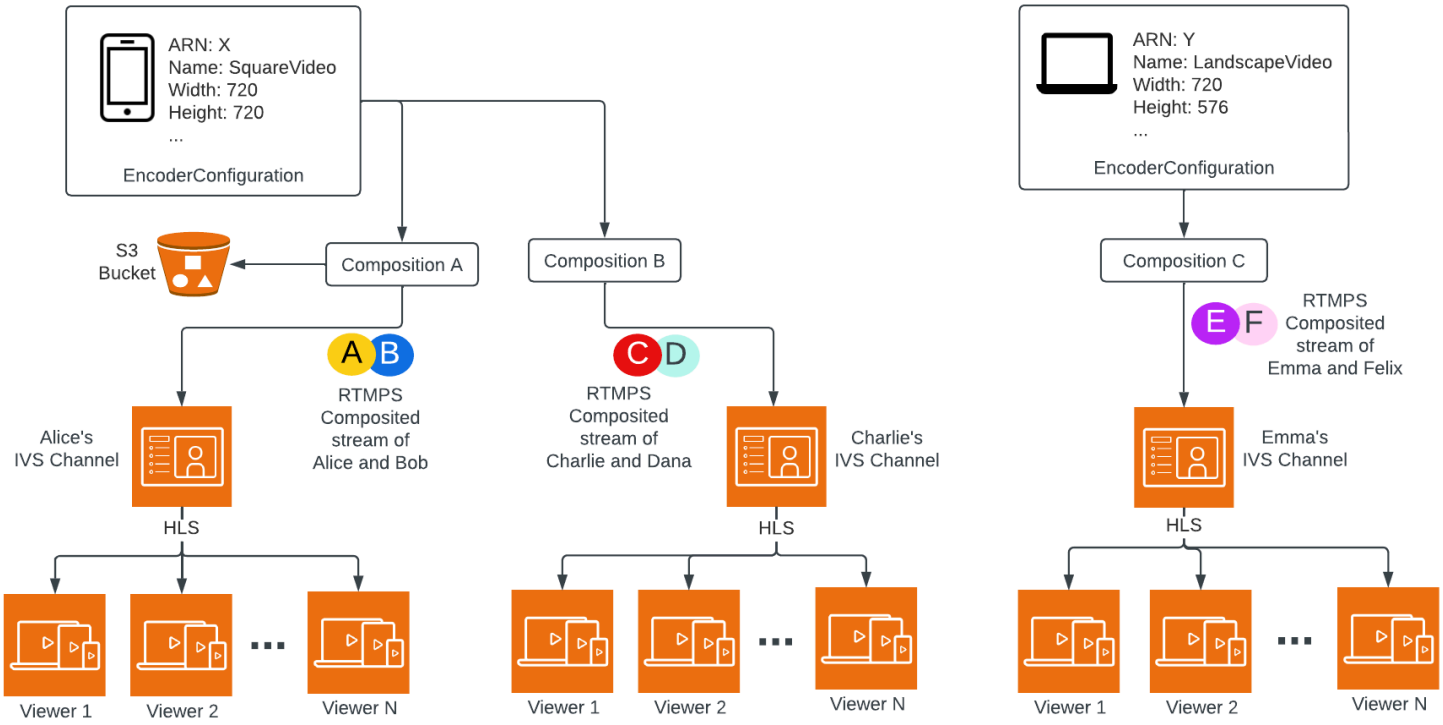
Como alternativa, para transmitir um palco para um canal IVS, você pode fazer a composição do lado do cliente. Consulte Habilitar vários hosts em um stream do IVS no Guia do usuário do streaming de baixa latência do IVS.

## API do IVS

A composição do servidor usa estes elementos-chave da API:

- Um objeto EncoderConfiguration permite que você personalize o formato do vídeo a ser gerado (altura, largura, taxa de bits e outros parâmetros de streaming). Você pode reutilizar um EncoderConfiguration sempre que chamar o endpoint StartComposition.
- Os endpoints de composição rastreiam a composição do vídeo e a saída para um canal do IVS.
- O StorageConfiguration rastreia o bucket do S3 no qual as composições são gravadas.

Para usar a composição do servidor, você precisa criar um EncoderConfiguration e anexá-lo ao chamar o endpoint StartComposition. Neste exemplo, o EncoderConfiguration do SquareVideo é usado em duas composições:

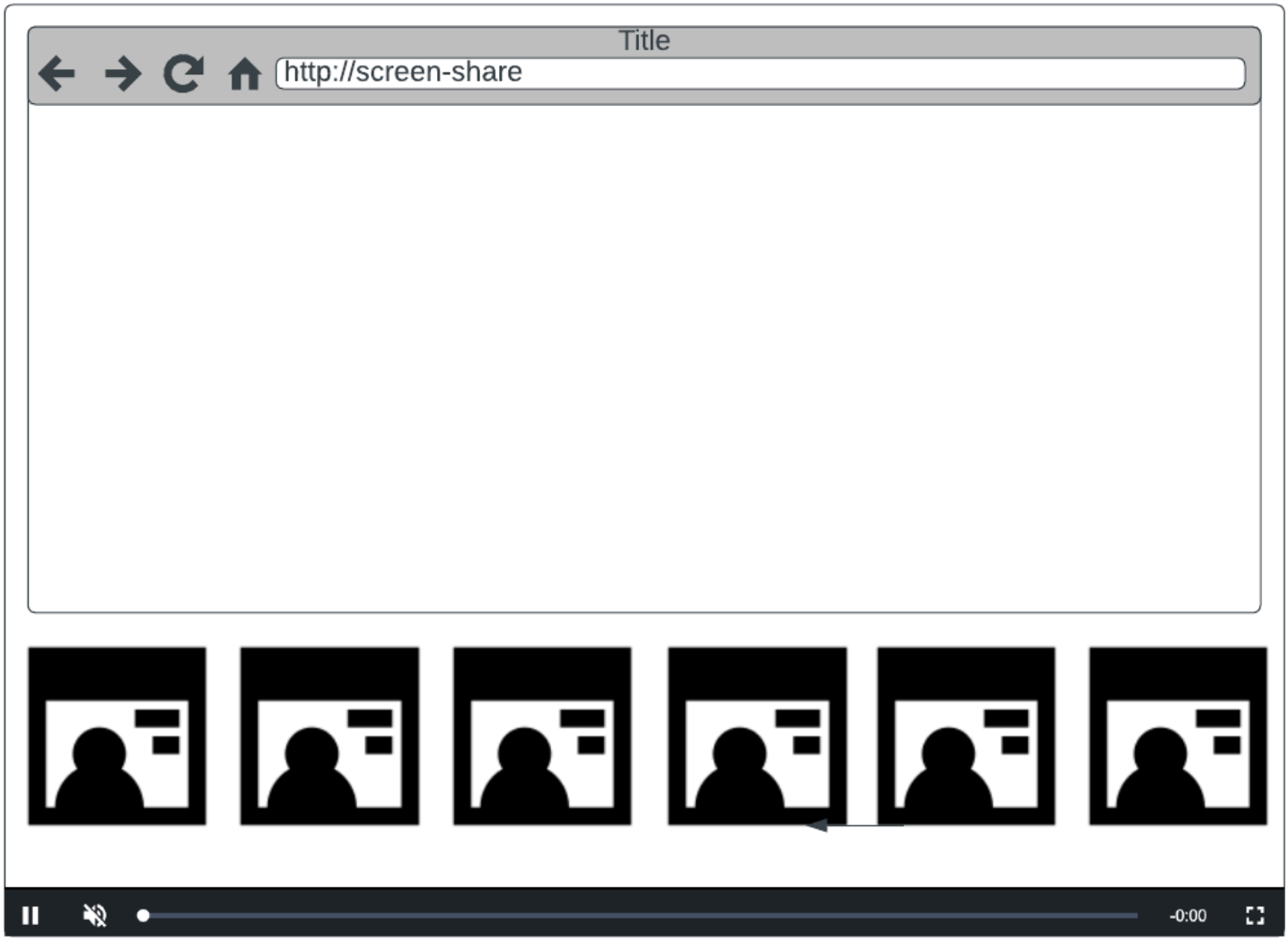



Para obter informações completas, consulte a [Referência de API da Transmissão em tempo real do IVS].

## Layouts

Por padrão, o recurso de composição do servidor usa um layout de grade para organizar os participantes do palco em espaços com tamanhos iguais:

Esse layout fornece uma opção para os clientes configurarem e invocarem um slot em destaque. O slot em destaque está na tela principal, com outros participantes mostrados abaixo em slots do mesmo tamanho:




Obs.: a resolução máxima suportada por um publicador de palco na composição do servidor é de 1080p. Se um publicador enviar um vídeo acima de 1080p, ele será renderizado como participante somente de áudio.

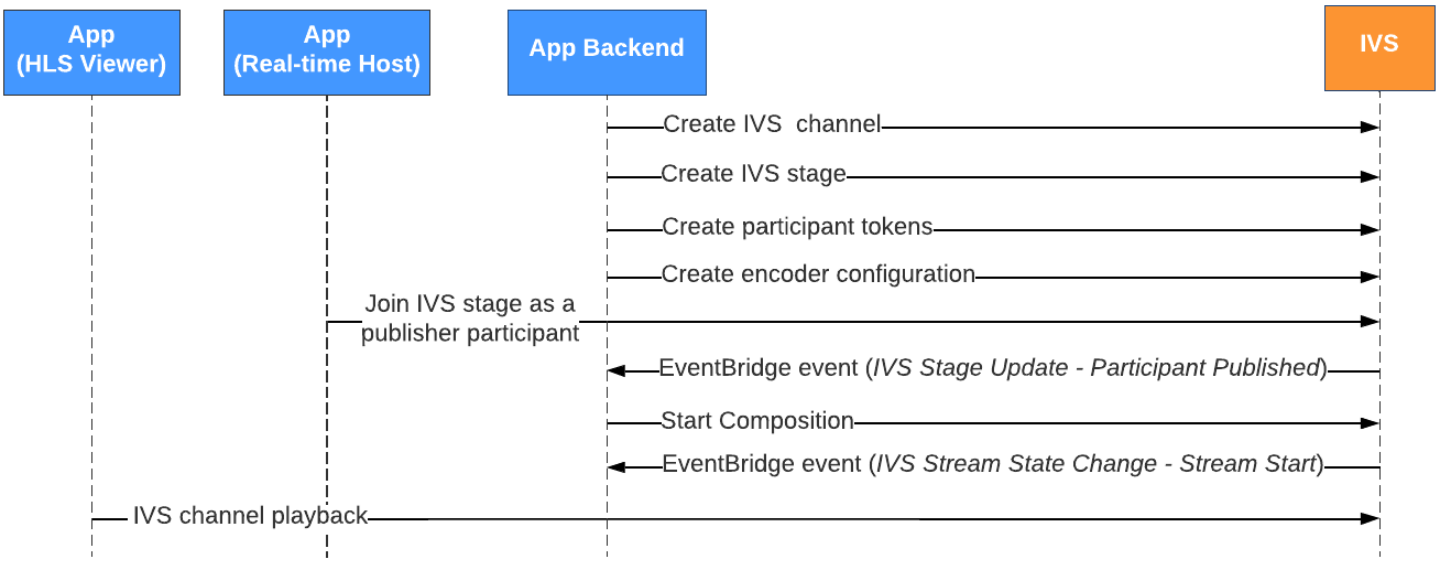
# Conceitos básicos

## Pré-requisitos

Para usar a composição do servidor, você deve ter um palco com publicadores ativos e usar um canal do IVS e/ou um bucket do S3 como destino da composição. Abaixo, descrevemos um possível fluxo de trabalho que usa eventos do EventBridge para iniciar uma composição que transmite o palco para um canal do IVS quando um participante publica. Como alternativa, você pode iniciar e interromper as composições com base na lógica da sua própria aplicação. Consulte Gravação

composta para ver outro exemplo que mostra o uso da composição do servidor para gravar um palco diretamente em um bucket do S3.

1. Crie um canal do IVS. Consulte Conceitos básicos do streaming de baixa latência do Amazon IVS.
2. Crie um palco do IVS e tokens de participante para cada publicador.
3. Crie um EncoderConfiguration.
4. Entre no palco e publique nele. (Consulte as seções “Publicação e inscrição” dos guias de SDK de transmissão de streaming em tempo real: Web, Android e iOS.)
5. Quando você receber um evento do EventBridge publicado pelo participante, chame StartComposition.
6. Aguarde alguns segundos e veja a visualização composta na reprodução do canal.




Obs.: uma composição será desligada automaticamente após 60 segundos de inatividade dos participantes do publicador no palco. Nesse ponto, a composição será encerrada e passará para um estado `STOPPED`. Uma composição será excluída automaticamente após alguns minutos no estado `STOPPED`.

## Instruções da CLI

Usar a AWS CLI é uma opção avançada e exige que você baixe e configure a CLI em sua máquina primeiro. Para obter mais detalhes, consulte o Guia do usuário da Interface de Linhas de Comando da AWS.

Agora é possível usar a CLI para criar e gerenciar recursos. Os endpoints da composição estão no namespace `ivs-realtime`.

## Criar o recurso EncoderConfiguration

Um objeto EncoderConfiguration permite que você personalize o formato do vídeo gerado (altura, largura, taxa de bits e outros parâmetros de streaming). Você pode reutilizar um EncoderConfiguration sempre que chamar o endpoint de composição, conforme explicado na próxima etapa.

O comando abaixo cria um recurso EncoderConfiguration que configura parâmetros de composição de vídeo do servidor, como taxa de bits, taxa de quadros e resolução do vídeo:

```
aws ivs-realtime create-encoder-configuration --name "MyEncoderConfig" --video
 "bitrate=2500000,height=720,width=1280,framerate=30"
```

A resposta é:

```
{
   "encoderConfiguration": {
      "arn": "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/9W59OBY2M8s4",
      "name": "MyEncoderConfig",
      "tags": {},
      "video": {
         "bitrate": 2500000,
         "framerate": 30,
         "height": 720,
         "width": 1280
      }
   }
}
```

## Iniciar uma composição

Usando o ARN do EncoderConfiguration fornecido na resposta acima, crie seu recurso de composição:

```
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"}}]'
```

A resposta mostrará que a composição foi criada com um estado `STARTING`. Quando a composição começa a publicar a composição, o estado passará para `ACTIVE`. (Você poderá ver o estado chamando o endpoint ListCompositions ou GetComposition.)

Quando uma composição for `ACTIVE`, a visualização composta do palco do IVS ficará visível no canal do IVS, usando ListCompositions:

```
aws ivs-realtime list-compositions
```

A resposta é:

```
{
   "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": {}
      }
   ]
}
```

Obs.: você precisa que os participantes do publicador publiquem ativamente no palco para manter a composição viva. Para obter mais informações, Consulte as seções “Publicação e inscrição” dos guias de SDK de transmissão de streaming em tempo real: Web, Android e iOS. Você deverá criar um token de palco distinto para cada participante.

## Habilitar o compartilhamento de tela

Para usar um layout fixo de compartilhamento de tela, siga as etapas abaixo.

## Criar o recurso EncoderConfiguration

O comando abaixo cria um recurso EncoderConfiguration que configura parâmetros de composição do servidor (taxa de bits, taxa de quadros e resolução do vídeo).

```
aws ivs-realtime create-encoder-configuration --name "test-ssc-with-screen-share" --
video={bitrate=2000000,framerate=30,height=720,width=1280}
```

Crie um token de participante do palco com um atributo `screen-share`. Como especificaremos `screen-share` como o nome do slot `featured`, será necessário criar um token de palco com o atributo `screen-share` definido como `true`:

```
aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes screen-share=true
```

A resposta é:

```
{
   "participantToken": {
      "attributes": {
         "screen-share": "true"
      },
      "expirationTime": "2023-08-04T05:26:11+00:00",
      "participantId": "E813MFklPWLF",
      "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2OTExMjY3NzEsImlhdCI6MTY5MTA4MzU3MSwianRpIjoiRT
   }
}
```

## Iniciar a composição

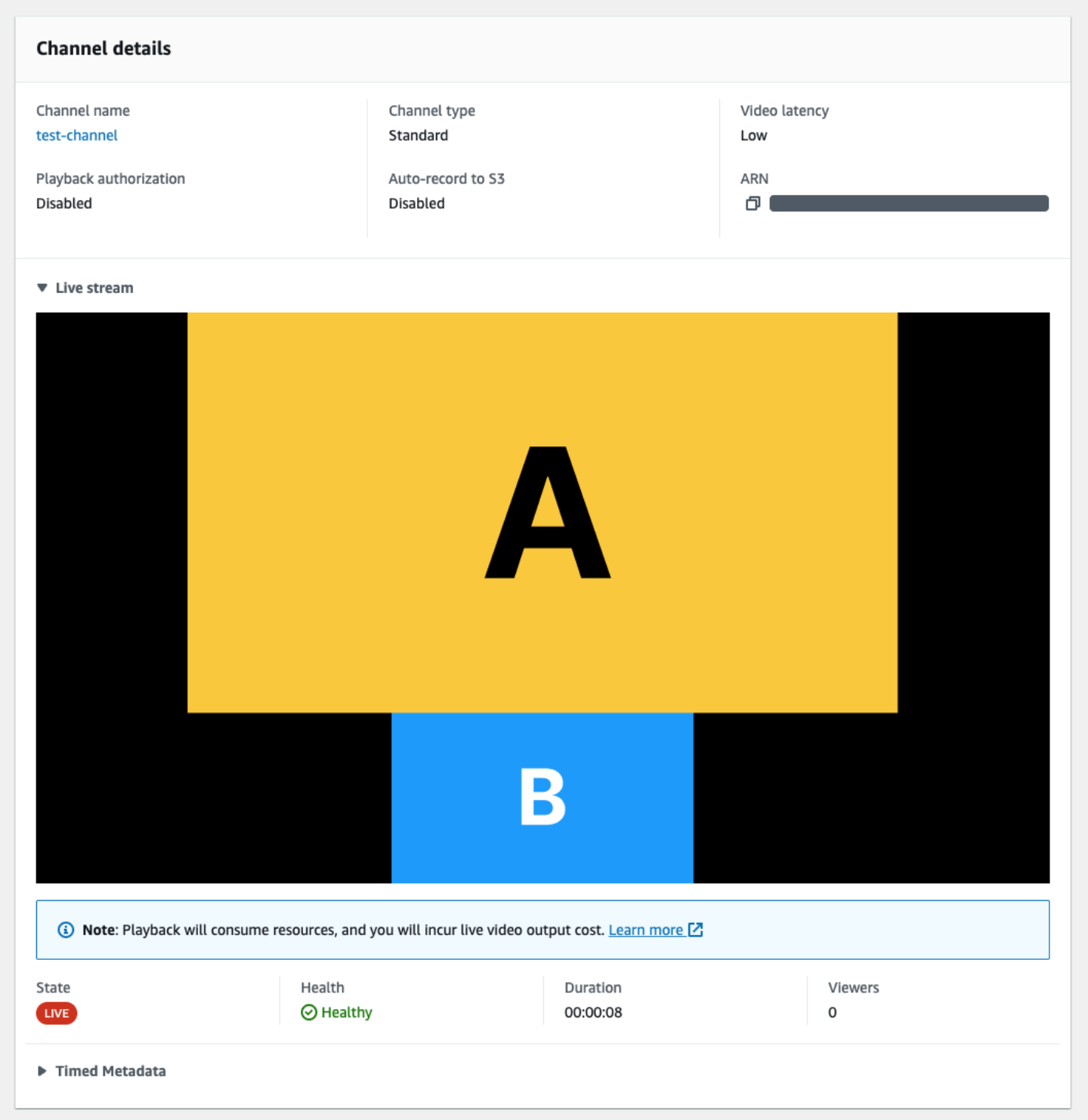
Para iniciar a composição usando o recurso de compartilhamento de tela, usaremos este comando:

```
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}"
```

A resposta é:

```
{
   "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" : { }
   }
}
```

Quando o participante `E813MFklPWLF` do palco ingressar no palco, o vídeo desse participante será exibido no espaço em destaque e todos os outros publicadores do palco serão renderizados abaixo do espaço:




## Parar a composição

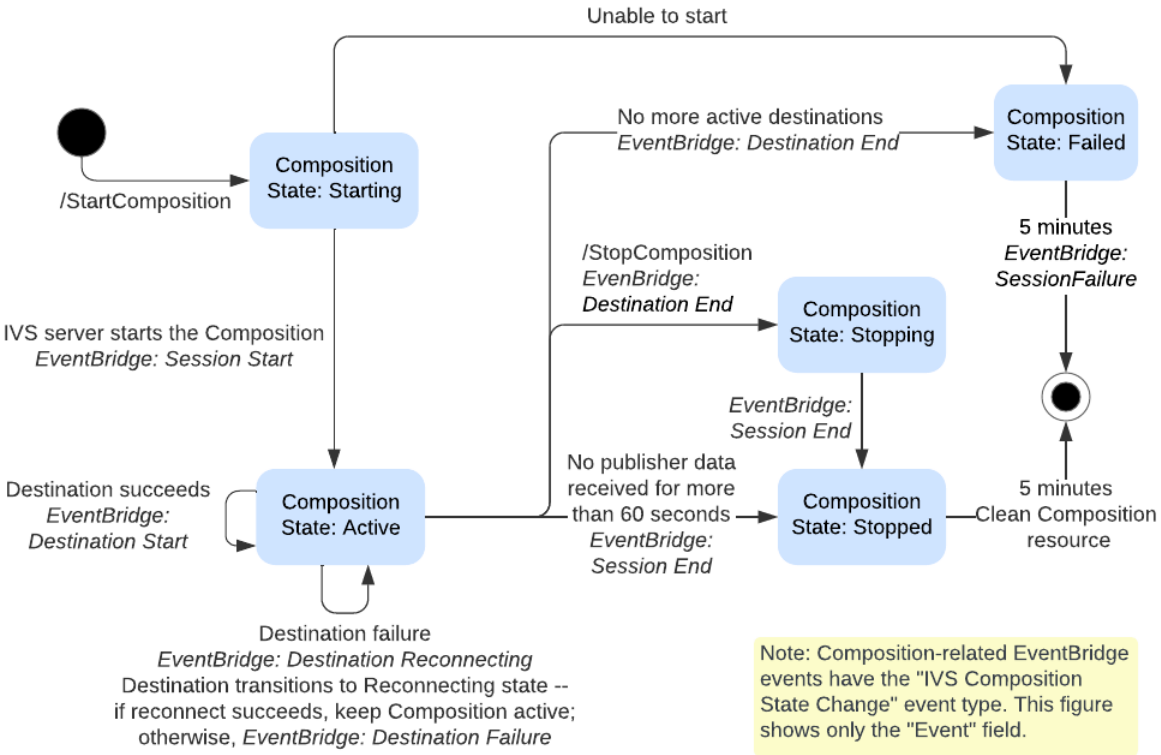
Para interromper uma composição em qualquer ponto, chame o endpoint StopComposition:

```
aws ivs-realtime stop-composition --arn arn:aws:ivs:us-east-1:927810967299:composition/
B19tQcXRgtoz
```

## Ciclo de vida da composição

Use o diagrama abaixo para entender as transições de estado de uma composição. Em alto nível, o ciclo de vida de uma composição é o seguinte:

1. Um recurso de composição é criado quando o usuário chama o endpoint StartComposition.
2. Depois que o IVS inicia a composição com sucesso, um evento “Alteração do estado da composição do IVS (início da sessão)” do EventBridge será enviado. Consulte Usar o EventBridge com o streaming em tempo real do IVS para obter detalhes sobre eventos.
3. Quando uma composição estiver em um estado ativo, o seguinte poderá acontecer:
   - O usuário interrompe a composição: se o endpoint StopComposition for chamado, o IVS iniciará um desligamento normal da composição, enviando eventos de “Fim de destino” seguidos por um evento de “Fim da sessão”.
   - A composição realiza o desligamento automático: se nenhum participante estiver publicando ativamente no palco do IVS, a composição será finalizada automaticamente após 60 segundos e os eventos do EventBridge serão enviados.
   - Falha no destino: se um destino falhar inesperadamente (por exemplo, o canal do IVS for excluído), o destino passará para o estado `RECONNECTING` e um evento de “Reconexão de destino” será enviado. Se a recuperação for impossível, o IVS fará a transição do destino para o estado `FAILED` e um evento de “Falha no destino” será enviado. O IVS mantém a composição viva se houver ao menos um de seus destinos ativo.
4. Quando a composição estiver no estado `STOPPED` ou `FAILED`, ela passará automaticamente por limpeza após cinco minutos. (Em seguida, ela não será mais recuperada por ListCompositions ou GetComposition.)


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.

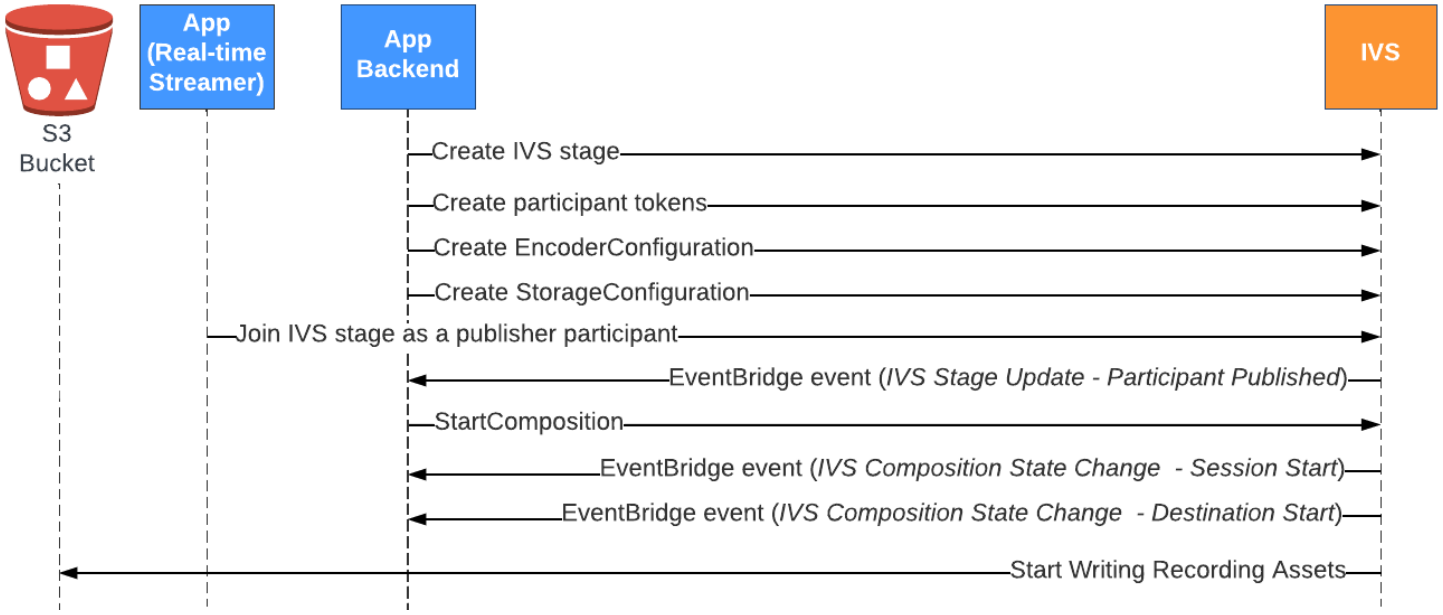
# Gravação composta (streaming em tempo real)

Este documento explica como usar o recurso de gravação composta na composição do servidor. A gravação composta permite gerar gravações HLS de um palco do IVS combinando efetivamente todos os publicadores de palco em uma visualização usando um servidor do IVS e salvando o vídeo resultante em um bucket do S3.

## Pré-requisitos

Para usar a gravação composta, você deve ter um estágio com editores ativos e um bucket S3 para usar como destino de gravação. Abaixo, descrevemos um possível fluxo de trabalho que usa EventBridge eventos para registrar uma composição em um bucket do S3. Como alternativa, você pode iniciar e interromper as composições com base na lógica da sua própria aplicação.

1. Crie um palco do IVS e tokens de participante para cada publicador.
2. Crie um EncoderConfiguration(um objeto representando como o vídeo gravado deve ser renderizado).
3. Crie um bucket S3 e um StorageConfiguration(onde o conteúdo da gravação será armazenado).
4. Entre no palco e publique nele.
5. Ao receber um EventBridge evento publicado pelo participante, ligue StartCompositioncom um DestinationConfiguration objeto do S3 como destino
6. Após alguns segundos, você verá os segmentos HLS como persistentes em seus buckets do S3.




Obs.: uma composição será desligada automaticamente após 60 segundos de inatividade dos participantes do publicador no palco. Nesse ponto, a composição será encerrada e passará para um estado `STOPPED`. Uma composição será excluída automaticamente após alguns minutos no estado `STOPPED`. Para obter detalhes, consulte Ciclo de vida da composição em Composição do servidor.

## Exemplo de gravação composta: StartComposition com um destino de bucket S3

O exemplo abaixo mostra uma chamada típica para o StartCompositionendpoint, especificando o S3 como o único destino para a composição. Depois que a composição for transferida para um estado `ACTIVE`, os segmentos de vídeo e os metadados começarão a ser gravados no bucket do S3 especificado pelo objeto `storageConfiguration`. Para criar composições com layouts diferentes, consulte “Layouts” em Composição do servidor e na Referência da API de Transmissão em tempo real do IVS.

### Solicitação

```
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"
}
```

## Resposta

```
{
    "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": {}
    }
}
```

O `recordingPrefix` campo, presente na StartComposition resposta, pode ser usado para determinar onde o conteúdo da gravação será armazenado.

## Conteúdo do registro

Quando a composição passar para um `ACTIVE` estado, você começará a ver segmentos de vídeo HLS e arquivos de metadados sendo gravados no bucket do S3 que foi fornecido durante a chamada. StartComposition Esses conteúdos estão disponíveis para pós-processamento ou reprodução como vídeo sob demanda.

Observe que após a ativação de uma composição, um evento “Mudança no estado da composição do IVS” será emitido e poderá levar algum tempo antes que os arquivos de manifesto e os segmentos de vídeo sejam gravados. Recomendamos que você reproduza ou processe transmissões gravadas somente após o recebimento do evento “Mudança no estado da composição do IVS (fim da sessão)”. Para obter detalhes, consulte Usando EventBridge com o IVS Real-Time Streaming.

Veja a seguir um exemplo de estrutura de diretório e conteúdo de uma gravação de uma sessão do IVS ao vivo:

```
MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/composite
   events
      recording-started.json
      recording-ended.json
   media
      hls
```

A pasta `events` contém os arquivos de metadados correspondentes ao evento de gravação. Os arquivos de metadados JSON são gerados quando a gravação é iniciada, termina com êxito ou termina com falhas:

- `events/recording-started.json`
- `events/recording-ended.json`
- `events/recording-failed.json`

Uma determinada pasta `events` vai conter `recording-started.json` e `recording-ended.json` ou `recording-failed.json`.

Elas contêm metadados relacionados à sessão gravada e seus formatos de saída. Os detalhes de JSON são fornecidos abaixo.

A pasta `media` contém o conteúdo de mídia compatível. A subpasta `hls` contém todos os arquivos de mídia e manifesto gerados durante a sessão de composição e pode ser reproduzida com o reprodutor do IVS. O manifesto HLS está localizado na pasta `multivariant.m3u8`.

## Política de bucket para StorageConfiguration

Quando um StorageConfiguration objeto é criado, o IVS terá acesso para gravar conteúdo no bucket S3 especificado. Esse acesso é concedido por meio de modificações na política de bucket do S3. Se a política do bucket for alterada de modo a remover o acesso do IVS, as gravações novas e contínuas falharão.

O exemplo abaixo mostra uma política de bucket do S3 que permite que o IVS grave no bucket do 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"
                }
            }
        }
    ]
}
```

## Arquivos de metadados de JSON

Os metadados estão em formato JSON. Eles contêm as seguintes informações:

| Campo | Tipo | Obrigatório | Descrição |
|---|---|---|---|
| `stage_arn` | string | Sim | ARN do palco que está sendo usado como origem da composição. |
| `media` | objeto | Sim | Objeto que contém os objetos enumerados de conteúdo de mídia disponível para essa gravação. Valores válidos: `"hls"`. |
| `hls` | objeto | Sim | Campo enumerado que descreve a saída do formato HLS da Apple. |
| `duration_ms` | inteiro | Condicional | Duração do conteúdo de HLS gravado em milissegundos. Isso só está disponível quando `recording_status` é `"RECORDING_ENDED"` ou `"RECORDING_ENDED_WITH_FAILURE"`. Se ocorreu uma falha antes de qualquer gravação ter sido feita, é 0. |

| Campo | Tipo | Obrigatório | Descrição |
| --- | --- | --- | --- |
| `path` | string | Sim | Caminho relativo do prefixo S3 onde o conteúdo de HLS é armazenado. |
| `playlist` | string | Sim | Nome do arquivo da lista de reprodução principal de HLS. |
| `renditions` | objeto | Sim | Matriz de representações (variante de HLS) de objetos de metadados . Há sempre pelo menos uma representação. |
| `path` | string | Sim | Caminho relativo do prefixo S3 em que o conteúdo de HLS é armazenado para essa versão. |
| `playlist` | string | Sim | Nome do arquivo da lista de reprodução de mídia para esta versão. |
| `resolution_height` | inteiro | Condicional | Altura de resolução de pixels do vídeo codificado. Isso só estará disponível quando a versão contiver uma faixa de vídeo. |
| `resolution_width` | inteiro | Condicional | Largura de resolução de pixels do vídeo codificado. Isso só estará disponível quando a versão contiver uma faixa de vídeo. |

| Campo | Tipo | Obrigatório | Descrição |
|---|---|---|---|
| `recording_ended_at` | string | Condicional | Carimbo de data/hora RFC 3339 UTC em que a gravação terminou. Isso só está disponível quando `recording_status` é `"RECORDING_ENDED"` ou `"RECORDING_ENDED_WITH_FAILURE"`.<br><br>`recording_started_at` e `recording_ended_at` são carimbos de data/hora quando esses eventos são gerados e podem não corresponder exatamente aos carimbos de data/hora do segmento de vídeo HLS. Para determinar com precisão a duração de uma gravação, use o campo `duration_ms`. |
| `recording_started_at` | string | Condicional | Carimbo de data/hora RFC 3339 UTC de quando a gravação foi iniciada. Isso não estará disponível quando `recording_status` for `RECORDING_START_FAILED`.<br><br>Consulte a observação acima para `recording_ended_at`. |
| `recording_status` | string | Sim | O status da gravação. Valores válidos: `"RECORDING_STARTED"`, `"RECORDING_ENDED"`, `"RECORDING_START_FAILED"`, `"RECORDING_ENDED_WITH_FAILURE"`. |

| Campo | Tipo | Obrigatório | Descrição |
|---|---|---|---|
| `recording_status_message` | string | Condicional | Informações descritivas sobre o status. Isso só está disponível quando `recording_status` é `"RECORDING_ENDED"` ou `"RECORDING_ENDED_WITH_FAILURE"` . |
| `version` | string | Sim | A versão do esquema de metadados. |

## Exemplo: 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
        }
      ]
    }
  }
}
```

## Exemplo: 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
        }
      ]
    }
  }
}
```

## Exemplo: 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
        }
      ]
```

```
    }
  }
}
```

# Reprodução de conteúdo gravado de buckets privados

Por padrão, o conteúdo gravado é privado. Portanto, esses objetos são inacessíveis para reprodução usando o URL direto do S3. Ao tentar abrir a lista de reprodução multivariada HLS (arquivo m3u8) para reprodução usando o Reprodutor do IVS ou outro player, você receberá mensagem de um erro (p. ex., “Você não tem permissão para acessar o recurso solicitado”). Em vez disso, você pode reproduzir esses arquivos com o Amazon CloudFront CDN (Content Delivery Network).

CloudFront as distribuições podem ser configuradas para fornecer conteúdo de buckets privados. Normalmente, isso é preferível a ter buckets de acesso aberto, onde as leituras ignoram os controles oferecidos por. CloudFront Você pode configurar sua distribuição para ser atendida a partir de um bucket privado criando um controle de acesso de origem (OAC), que é um CloudFront usuário especial que tem permissões de leitura no bucket de origem privado. Você pode criar o OAC depois de criar sua distribuição, por meio do CloudFront console ou da API. Consulte Criação de um novo controle de acesso de origem no Amazon CloudFront Developer Guide.

## Configurando a reprodução usando CloudFront com o CORS ativado

Este exemplo mostra como um desenvolvedor pode configurar uma CloudFront distribuição com o CORS ativado, permitindo a reprodução de suas gravações de qualquer domínio. Isso é especialmente útil durante a fase de desenvolvimento, mas você pode modificar o exemplo abaixo para atender às suas necessidades de produção.

### Etapa 1: Crie um bucket do S3

Crie um bucket do S3 que será usado para armazenar as gravações. Observe que o bucket precisa estar na mesma região que você usa para seu fluxo de trabalho do IVS.

Adicione uma política CORS permissiva ao bucket:

1. No console da AWS, acesse a guia Permissões de bucket do S3.
2. Copie a política do CORS abaixo e cole-a em Compartilhamento de recursos de origem cruzada (CORS). Isso habilitará o acesso ao CORS no bucket do S3.

```
[
```

```
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

## Etapa 2: criar uma CloudFront distribuição

Consulte Criação de uma CloudFront distribuição no Guia do CloudFront desenvolvedor.

No Console da AWS, insira as seguintes informações:

| Para este campo… | Escolha isto... |
| --- | --- |
| Domínio de origem | O bucket do S3 que você criou na etapa anterior |
| Acesso de origem | Configurações de controle de acesso de origem (recomendado) usando os parâmetros padrão |
| Comportamento padrão do cache: política de protocolo do visualizador | Redirect HTTP to HTTPS |
| Comportamento padrão do cache: métodos HTTP permitidos | GET, HEAD e OPTIONS |

| Para este campo… | Escolha isto... |
| --- | --- |
| Comportamento padrão do cache: chaves de cache e solicitações de origem | CachingDisabled política |
| Comportamento padrão do cache: política de solicitação de origem | CORS-S3Origin |
| Comportamento padrão do cache: política de cabeçalhos de resposta | SimpleCORS |
| Firewall da aplicação Web | Habilitar as proteções de segurança |

Em seguida, salve a CloudFront distribuição.

### Etapa 3: configurar a política de bucket do S3

1. Exclua tudo StorageConfiguration o que você configurou para o bucket do S3. Isso removerá todas as políticas de bucket que foram adicionadas automaticamente ao criar a política para esse bucket.
2. Acesse sua CloudFront distribuição, certifique-se de que todos os campos de distribuição estejam nos estados definidos na etapa anterior e copie a política do bucket (use o botão Copiar política).
3. Acesse seu bucket do S3. Na guia Permissões, selecione Editar política de bucket e cole a política de bucket que copiou na etapa anterior. Após essa etapa, a política de bucket deve ter a CloudFront política exclusivamente.
4. Crie um StorageConfiguration, especificando o bucket do S3.

Depois que o StorageConfiguration for criado, você verá dois itens na política de bucket do S3, um permitindo CloudFront a leitura do conteúdo e outro permitindo que o IVS grave o conteúdo. Um exemplo de uma política de bucket final, com CloudFront acesso IVS, é mostrado em Exemplo: Política de bucket do S3 com CloudFront acesso IVS.

### Etapa 4: reproduzir gravações

Depois de configurar com sucesso a CloudFront distribuição e atualizar a política do bucket, você poderá reproduzir gravações usando o reprodutor IVS:

1. Inicie uma composição bem-sucedida e verifique se você tem uma gravação armazenada no bucket do S3.
2. Depois de seguir as etapas 1 a 3 neste exemplo, os arquivos de vídeo devem estar disponíveis para consumo por meio do CloudFront URL. Seu CloudFront URL é o nome do domínio de distribuição na guia Detalhes no CloudFront console da Amazon. Será algo do tipo:

   `a1b23cdef4ghij.cloudfront.net`
3. Para reproduzir o vídeo gravado por meio da CloudFront distribuição, encontre a chave do objeto `multivariant.m3u8` do seu arquivo no bucket s3. Será algo do tipo:

   `FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/multivariant.m3u8`
4. Anexe a chave do objeto ao final do seu CloudFront URL. O URL final será mais ou menos assim:

   `https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/multivariant.m3u8`
5. Agora, você pode adicionar o URL final ao atributo de origem de um Reprodutor do IVS para assistir à gravação completa. Para assistir ao vídeo gravado, você pode usar a demonstração em Introdução no SDK do Reprodutor do IVS: guia da Web.

## Exemplo: Política de bucket do S3 com CloudFront acesso IVS

O trecho abaixo ilustra uma política de bucket do S3 que CloudFront permite ler conteúdo no bucket privado e o IVS gravar conteúdo no bucket. Obs.: não copie e cole o trecho abaixo em seu próprio bucket. Sua política deve conter os IDs relevantes para sua CloudFront distribuição StorageConfiguration e.

```
{
  "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"
        }
      }
    }
  ]
}
```

# Solução de problemas

- A composição não é gravada no bucket do S3 — verifique se o bucket e os StorageConfiguration objetos do S3 foram criados e estão na mesma região. Além disso, certifique-se de que o IVS tenha acesso ao bucket verificando sua política de bucket; consulte a Política de bucket para StorageConfiguration.
- Não consigo encontrar uma composição quando estou tocando ListCompositions— as composições são recursos efêmeros. Após passarem para o estado final, elas serão excluídos automaticamente após alguns minutos.
- Minha composição para automaticamente: uma composição será interrompida automaticamente se não houver um publicador no palco por mais de 60 segundos.

# Problema conhecido

A playlist de mídia escrita por gravação composta terá a tag `#EXT-X-PLAYLIST-TYPE:EVENT` enquanto a composição estiver em andamento. Quando a composição for concluída, a tag será atualizada para `#EXT-X-PLAYLIST-TYPE:VOD`. Para uma experiência tranquila de reprodução, recomendamos que você use essa lista de reprodução somente depois que a composição for finalizada com sucesso.

# Suporte para OBS e WHIP (streaming em tempo real)

Este documento explica como usar codificadores compatíveis com WHIP, como OBS, para publicar no streaming em tempo real do IVS. O WHIP (WebRTC-HTTP Ingestion Protocol) é um rascunho do IETF desenvolvido para padronizar a ingestão de WebRTC.

O WHIP permite a compatibilidade com softwares como o OBS, oferecendo uma alternativa (ao SDK de transmissão do IVS) para editoração eletrônica. Streamers mais sofisticados familiarizados com o OBS podem preferi-lo por seus recursos avançados de produção, como transições de cena, mixagem de áudio e gráficos de sobreposição. Isso fornece aos desenvolvedores uma opção versátil: usar o SDK de transmissão na web do IVS para publicação direta no navegador ou permitir que os streamers usem o OBS em seus desktops para obter ferramentas mais poderosas.

Além disso, o WHIP é benéfico em situações em que o uso do SDK de transmissão do IVS não é viável ou preferido. Por exemplo, em configurações que envolvem codificadores de hardware, o SDK de transmissão IVS pode não ser uma opção. No entanto, se o codificador suportar WHIP, você ainda poderá publicar diretamente do codificador no IVS.

## Guia OBS

O OBS suporta WHIP a partir da versão 3.0. Para começar, baixe o OBS v30 ou mais recente: https://obsproject.com/.

Para publicar em um estágio IVS usando OBS via WHIP, siga estas etapas:

1. Gere um token de participante com capacidade de publicação. Em termos de WHIP, um token de participante é um token de portador. Por padrão, os tokens de participantes expiram em 12 horas, mas você pode estender a duração em até 14 dias.
2. Clique em Settings (Configurações). Na seção Stream do painel Configurações, selecione WHIP no menu suspenso Serviço.
3. Para o Servidor, digite https://global.whip.live-video.net/.
4. Para o Token do Portador, insira o token do participante que você gerou na etapa 2.
5. Defina suas configurações de vídeo como faria normalmente, com algumas restrições:
   a. O streaming em tempo real IVS suporta entrada de até 720p a 8,5 Mbps. Se você exceder qualquer um desses limites, seu stream será desconectado.

b. Recomendamos definir o intervalo do quadro-chave no painel Saída para 1s ou 2s. Um intervalo baixo de quadros-chave permite que a reprodução do vídeo comece mais rapidamente para os espectadores. Também recomendamos definir a predefinição de uso da CPU para ultrarrápida e a configuração para latência zero, para permitir a menor latência.

c. Como o OBS não suporta transmissão simultânea, recomendamos manter sua taxa de bits abaixo de 2,5 Mbps. Isso permite que os espectadores em conexões de baixa largura de banda assistam.

6. Pressione Iniciar transmissão.

# Service Quotas (Streaming em tempo real)

A seguir estão os limites e as service quotas para endpoints, recursos e outras operações em tempo real do Amazon Interactive Video Service (IVS). As service quotas (também conhecidas como limites) correspondem ao número máximo de recursos ou operações de serviço para sua conta da AWS. Ou seja, esses limites são por conta da AWS, a menos que indicado de outra forma na tabela. Consulte também AWS Service Quotas.

Você usa um endpoint para se conectar programaticamente a um serviço da AWS. Consulte também AWS Service Endpoints.

Todas as cotas são aplicadas por região.

## Aumentos de cota de serviço

Para cotas que são ajustáveis, é possível solicitar um aumento de taxa por meio do Console da AWS. Use o console para também visualizar informações sobre cotas de serviço.

As cotas para taxa de chamadas da API não são ajustáveis.

## Cotas de taxa de chamada de API

| Tipo de endpoint | Endpoint | Padrão |
|---|---|---|
| Composição | `GetComposition` | 5 TPS |
| Composição | `ListCompositions` | 5 TPS |
| Composição | `StartComposition` | 5 TPS |
| Composição | `StopComposition` | 5 TPS |
| MediaEncoder | `CreateEncoderConfiguration` | 5 TPS |
| MediaEncoder | `DeleteEncoderConfiguration` | 5 TPS |
| MediaEncoder | `GetEncoderConfiguration` | 5 TPS |
| MediaEncoder | `ListEncoderConfigurations` | 5 TPS |

| Tipo de endpoint | Endpoint | Padrão |
|---|---|---|
| Estágio | `CreateParticipantToken` | 50 TPS |
| Estágio | `CreateStage` | 5 TPS |
| Estágio | `DeleteStage` | 5 TPS |
| Estágio | `DisconnectParticipant` | 5 TPS |
| Estágio | `GetParticipant` | 5 TPS |
| Estágio | `GetStage` | 5 TPS |
| Estágio | `GetStageSession` | 5 TPS |
| Estágio | `ListStages` | 5 TPS |
| Estágio | `UpdateStage` | 5 TPS |
| Estágio | `ListParticipants` | 5 TPS |
| Estágio | `ListParticipantEvents` | 5 TPS |
| Estágio | `ListStageSessions` | 5 TPS |
| StorageConfiguration | `CreateStorageConfiguration` | 5 TPS |
| StorageConfiguration | `DeleteStorageConfiguration` | 5 TPS |
| StorageConfiguration | `GetStorageConfiguration` | 5 TPS |
| StorageConfiguration | `ListStorageConfigurations` | 5 TPS |
| Tags | `ListTagsForResource` | 10 TPS |
| Tags | `TagResource` | 10 TPS |
| Tags | `UntagResource` | 10 TPS |

# Outras cotas

| Recurso ou atributo | Padrão | Ajustável | Descrição |
| --- | --- | --- | --- |
| EncoderConfigurations | 20 | Sim | Número máximo de recursos de configuração do codificador por conta. |
| Destinos de composição | 2 | Não | Número máximo de objetos de destino em um recurso de composição. |
| Composição: duração máxima | 24 | Não | A quantidade máxima de tempo que uma composição pode existir em horas. |
| Composições | 5 | Sim | Máximo de recursos simultâneos de composição por conta. |
| Duração da publicação ou inscrição do participante | 24 | Não | Período máximo que um participante pode realizar publicações ou permanecer inscrito em um palco, em horas. |
| Resolução da publicação do participante | 720p | Não | Resolução máxima dos vídeos publicados pelos participantes. |
| Taxa de bits de download do participante | 8.5 Mbps | Não | Taxa máxima de bits de download agregada em todas as inscrições de um participante. |
| Participantes do palco (editores) | 12 | Não | Número máximo de participantes que podem publicar em um palco ao mesmo tempo. |
| Participantes do palco (inscritos) | 10.000 | Sim | Número máximo de participantes que podem inscrever- |

| Recurso ou atributo | Padrão | Ajustável | Descrição |
| --- | --- | --- | --- |
| | | | se em um palco ao mesmo tempo. |
| Palcos | 100 | Yes | Número máximo de palcos por região da AWS. |

# Otimizações de streaming em tempo real

Para garantir que seus usuários tenham a melhor experiência possível ao transmitir e visualizar vídeos usando o streaming em tempo real do IVS, existem diversas maneiras de aprimorar ou otimizar partes da experiência usando os recursos que oferecemos hoje.

## Introdução

Ao otimizar a qualidade da experiência de um usuário, é importante considerar a experiência desejada, que pode mudar dependendo do conteúdo que está sendo assistido e das condições da rede.

Ao longo deste guia, focamos nos usuários que são publicadores de streams ou inscritos em streams, e consideramos as ações e as experiências desejadas por esses usuários.

## Streaming adaptável: codificação em camadas com a transmissão simultânea

Esse recurso é compatível somente nas seguintes versões do cliente:

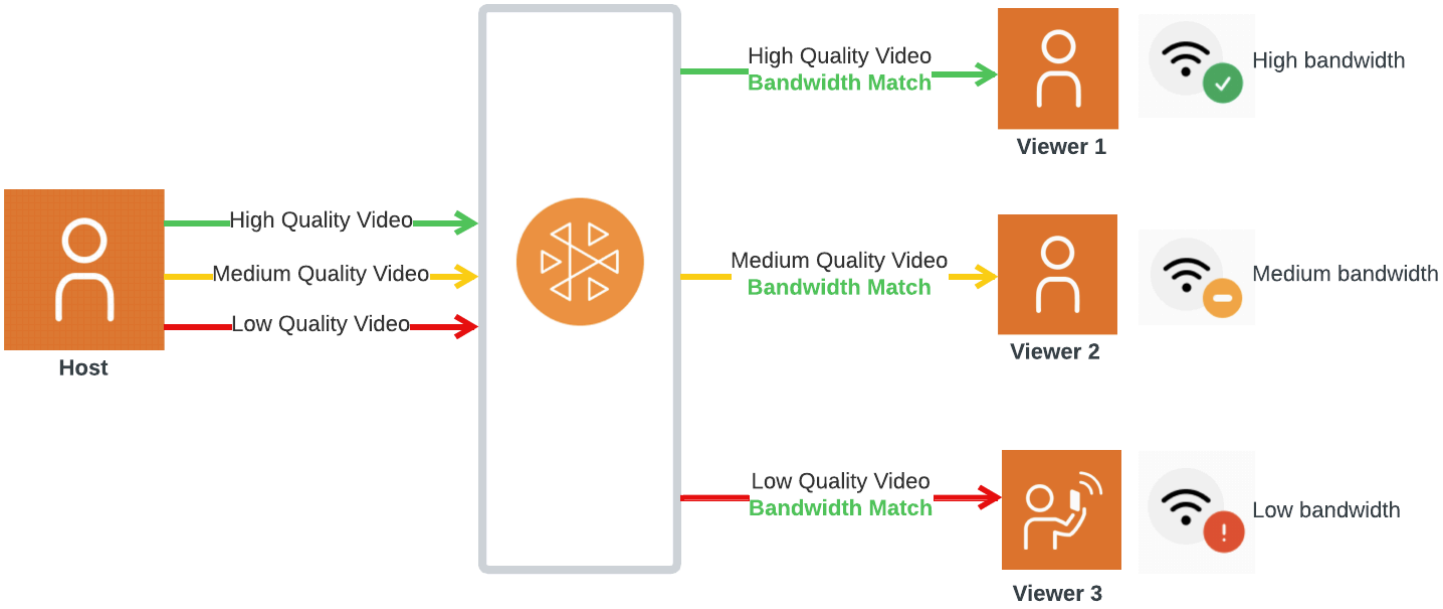
- iOS e Android 1.12.0 +
- Web 1.5.1 +

Você deve enviar um e-mail para amazon-ivs-simulcast@amazon .com para optar por esse recurso em sua conta. Habilitar a transmissão simultânea por meio da configuração do SDK não terá efeito, a menos que você tenha optado por participar.

Após ter optado pelo recurso, os publicadores codificarão várias camadas de vídeo e os inscritos se adaptarão ou alterarão automaticamente para a qualidade mais adequada para suas redes ao usar SDKs de transmissão em tempo real do IVS. Nós chamamos isso de codificação em camadas com a transmissão simultânea.

A codificação em camadas com a transmissão simultânea é compatível com Android e iOS e em navegadores de área de trabalho Chrome (para Windows e macOS). Não oferecemos suporte à codificação em camadas em outros navegadores.

No diagrama abaixo, o host está enviando três qualidades de vídeo (nomeadamente: alta, média e baixa). O IVS encaminha o vídeo da mais alta qualidade para cada espectador com base na largura de banda disponível, fornecendo uma experiência ideal para cada espectador. Se a conexão de rede do Espectador 1 for alterada de boa para ruim, o IVS automaticamente começa a enviar vídeo de qualidade inferior ao Espectador 1, para que ele possa continuar assistindo à transmissão ininterruptamente (com a melhor qualidade possível).




## Camadas, qualidades e taxas de quadros padrão

As qualidades e as camadas padrão fornecidas para usuários de dispositivos móveis e da Web são as seguintes:

| Dispositivos móveis (Android e iOS) | Web (Chrome) |
| --- | --- |
| Camada alta (ou personalizada):<br>• Taxa de bits máxima: 900.000 bps<br>• Taxa de quadros: 15 fps<br>• Resolução: 360x640 | Camada alta (ou personalizada):<br>• Taxa de bits máxima: 1.700.000 bps<br>• Taxa de quadros: 30 fps<br>• Resolução: 1280x720 |
| Camada média: nenhuma (não é necessária, pois a diferença entre as taxas de bits da camada alta e baixa em dispositivos móveis é pequena) | Camada média:<br>• Taxa de bits máxima: 700.000 bps<br>• Taxa de quadros: 20 fps |

| Dispositivos móveis (Android e iOS) | Web (Chrome) |
| --- | --- |
| | • Resolução: 640x360 |
| Camada baixa:<br>• Taxa de bits máxima: 150.000 bps<br>• Taxa de quadros: 15 fps<br>• Resolução: 180x320 | Camada baixa:<br>• Taxa de bits máxima: 200.000 bps<br>• Taxa de quadros: 15 fps<br>• Resolução: 320x180 |

## Configuração da codificação em camadas com a transmissão simultânea

Para usar a codificação em camadas com transmissão simultânea, você deve ter optado pelo recurso e habilitado isso no cliente. Se você ativá-lo, verá um aumento na taxa de bits geral transmitida, com o benefício de menos congelamento de vídeo.

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
```

# Configurações de transmissão

Esta seção explora outras configurações que você pode fazer em seus fluxos de vídeo e áudio.

## Alterar taxa de bits do fluxo

Para alterar a taxa de bits do fluxo de vídeo, use os exemplos de configuração a seguir.

### 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
```

## Alterar da taxa de quadros do fluxo de vídeo

Para alterar a taxa de quadros do fluxo de vídeo, use os exemplos de configuração a seguir.

### 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
```

## Otimização da taxa de bits de áudio e compatibilidade com estéreo

Para alterar a taxa de bits e as configurações do estéreo, use os exemplos de configuração a seguir.

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
```

# Otimizações sugeridas

| Cenário | Recomendações |
| --- | --- |
| Fluxos com texto ou conteúdo em movimento lento, como apresentações ou slides | Use codificação em camadas com transmissão simultânea ou configure fluxos com menor taxa de quadros. |
| Transmissões com ação ou muito movimento | Use codificação em camadas com transmissão simultânea. |
| Transmissões com conversa ou pouco movimento | Use a codificação em camadas com transmissão simultânea ou escolha somente áudio (consulte “Inscrição em participantes” nos guias do SDK de Transmissão em tempo real: Web, Android e iOS). |

| Cenário | Recomendações |
| --- | --- |
| Usuários fazendo streaming com dados limitados | Use codificação em camadas com transmissão simultânea ou, se quiser menor uso de dados para todos, configure uma taxa de quadros mais baixa e diminua a taxa de bits manualmente. |

# Recursos e suporte (Streaming em tempo real)

## Recursos

https://ivs.rocks/ é um site dedicado para navegar pelo conteúdo publicado (demonstrações, amostras de código, publicações de blog), calcular o custo e experimentar o Amazon IVS com demonstrações ao vivo.

## Demonstrações


A demonstração do streaming em tempo real do IVS para iOS e Android apresenta aos desenvolvedores como usar o Amazon IVS para desenvolver uma aplicação atrativa de conteúdo social em tempo real gerado pelo usuário. Esta aplicação apresenta um feed deslizável de streams em tempo real geradas pelo usuário. Os usuários podem criar streams de vídeo e salas somente de áudio. Os convidados do stream de vídeo podem ingressar no modo de convidado ou versus

(VS). As instruções sobre como implantar o back-end necessário e desenvolver a aplicação estão disponíveis nos seguintes repositórios do 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/

# Suporte

O AWS Support Center disponibiliza uma variedade de planos que fornecem acesso a ferramentas e experiência para oferecer suporte às suas soluções da AWS. Todos os planos de suporte oferecem acesso 24 horas por dia, 7 dias por semana ao atendimento ao cliente. Para obter suporte técnico e mais recursos para planejar, implantar e aprimorar seu ambiente da AWS, selecione o plano de suporte que melhor se alinhe ao seu caso de uso da AWS.

O AWS Premium Support é um canal de suporte de resposta rápida com atendimento individual cujo objetivo é ajudar você a desenvolver e executar aplicações na AWS.

O AWS re:Post é um site de perguntas e respostas baseado na comunidade para desenvolvedores discutirem questões técnicas relacionadas ao Amazon IVS.

Entre em contato conosco contém links para consultas sobre sua conta ou faturamento. Para dúvidas técnicas, use os fóruns de discussão ou links de suporte acima.

# Glossário

Consulte também o Glossary da AWS. Na tabela abaixo, LL significa streaming de baixa latência do IVS; RT, streaming em tempo real do IVS.

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| AAC | Codificação de áudio avançado. O AAC é um padrão de codificação de áudio para compressão de áudio digital com perdas. Projetado para ser o sucessor do formato MP3, o AAC geralmente alcança uma qualidade de som superior ao MP3 com a mesma taxa de bits. O AAC foi padronizado pela ISO e pela IEC como parte das especificações MPEG-2 e MPEG-4. | ✓ | ✓ | |
| Streaming com taxa de bits adaptável | O streaming com taxa de bits adaptável (ABR) permite que o reprodutor do IVS mude para uma taxa de bits mais baixa quando a qualidade da conexão piora e volte para uma taxa de bits mais alta quando a qualidade da conexão melhora. | ✓ | | |
| Streaming adaptável | Consulte Codificação em camadas com a transmissão simultânea. | | ✓ | |
| Usuário administrativo | Um usuário da AWS com acesso administrativo aos recursos e serviços disponíveis em uma conta da AWS. Consulte Terminologia no Guia do usuário da configuração da AWS. | ✓ | ✓ | ✓ |
| ARN | Nome do recurso da Amazon, um identificador exclusivo de um recurso da AWS. Os formatos específicos do ARN dependem do tipo do recurso. Para conhecer os formatos de ARN usados pelos recursos do IVS, consulte a Referência de autorização do serviço. | ✓ | ✓ | ✓ |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Taxa de proporção | Descreve a proporção entre a largura e a altura do quadro. Por exemplo, 16:9 é a proporção que corresponde à resolução Full HD ou 1080p. | ✓ | ✓ | |
| Modo de áudio | Uma configuração de áudio predefinida ou personalizada otimizada para diferentes tipos de usuários de dispositivos móveis e o equipamento que eles usam. Consulte SDK de Transmissão do IVS: modos de áudio móvel (streaming em tempo real). | | ✓ | |
| AVC, H.264, MPEG-4 Parte 10 | Codificação de vídeo avançado, também conhecida como H.264 ou MPEG-4 Parte 10, um padrão de compressão para vídeo digital com perdas. | ✓ | ✓ | |
| Substituição de plano de fundo | Um tipo de filtro de câmera que permite que criadores de fluxo ao vivo alterem seus planos de plano de fundo. Consulte Substituição de plano de fundo em SDK de Transmissão do IVS: filtros de câmera de terceiros (streaming em tempo real). | | ✓ | |
| Taxa de bits | Uma métrica de streaming para o número de bits transmitidos ou recebidos por segundo. | ✓ | ✓ | |
| Transmissão, transmissores | São outros termos para fluxo, streamer. | ✓ | | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Armazenamento em buffer | Uma condição que ocorre quando o dispositivo de reprodução não consegue baixar o conteúdo antes que ele deva ser reproduzido. O armazenamento em buffer pode se manifestar de várias maneiras: o conteúdo pode parar e começar aleatoria mente (também conhecido como engasgo), o conteúdo pode parar por longos períodos (também conhecido como congelamento) ou o reprodutor do IVS pode pausar a reprodução. | ✓ | ✓ | |
| Lista de reproduçã o de intervalo de bytes | Uma lista de reprodução mais granular do que a lista de reprodução HLS padrão. A lista de reprodução HLS padrão é composta de arquivos de mídia de dez segundos. Com uma lista de reprodução de intervalo de bytes, a duração do segmento é a mesma do intervalo de quadros-c have configurado para o fluxo.<br><br>A lista de reprodução com intervalo de bytes está disponível somente para as transmissões que foram gravadas automaticamente em um bucket do S3. Ela é criada além da lista de reprodução HLS. Consulte Listas de reprodução de intervalo de bytes em Gravação automática no Amazon S3 (streaming de baixa latência). | ✓ | | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| CBR | Taxa de bits constante, um método de controle de taxa para codificadores que mantém uma taxa de bits consistente durante toda a reprodução de um vídeo, independentemente do que esteja acontecendo durante a transmissão. As pausas na ação podem ser preenchidas para atingir a taxa de bits desejada e os picos podem ser quantizados pelo ajuste da qualidade da codificação para corresponder à taxa de bits desejada. É altamente recomendável usar CBR em vez de VBR. | ✓ | ✓ | |
| CDN | Rede de entrega de conteúdo ou Rede de distribuição de conteúdo, uma solução distribuída geograficamente que otimiza a entrega de conteúdo, como streaming de vídeo, ao aproximá-lo de onde os usuários estão localizados. | ✓ | | |
| Channel (Canal) | Um recurso do IVS que armazena a configuração para streaming, incluindo um servidor de ingestão, uma chave de fluxo, um URL de reprodução e opções de gravação. Os streamers usam a chave de stream associada a um canal para iniciar uma transmissão. Todas as métricas e eventos gerados durante uma transmissão são associados a um recurso de canal. | ✓ | | |
| Tipo de canal | Determina a resolução e a taxa de quadros permitidas para o canal. Consulte Channel Types em IVS Low-Latency Streaming API Reference. | ✓ | | |
| Registro em log de chat | Uma opção avançada que pode ser habilitada pela associação de uma configuração de registro em log com uma sala de chat. | | | ✓ |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Sala de chat | Um recurso do IVS que armazena a configuração de uma sessão de conversa, incluindo recursos opcionais, como Processador de análise de mensagens e Registro em log de chat. Consulte Step 2: Create a Chat Room em Getting Started with IVS Chat. | | | ✓ |
| Composição do cliente | Usa um dispositivo host para mixar fluxos de áudio e vídeo dos participantes do palco e depois os envia como um fluxo composto para um canal do IVS. Isso permite mais controle sobre o aspecto da composição à custa de uma maior utilização dos recursos do cliente e de um risco maior de que um problema em um palco ou em um host afete os espectadores.<br><br>Consulte também composição do servidor. | ✓ | ✓ | |
| CloudFront | Um serviço de CDN fornecido pela Amazon. | ✓ | | |
| CloudTrail | Um serviço da AWS para coletar, monitorar, analisar e reter eventos e atividades da conta da AWS e de fontes externas. Consulte Registro de chamadas de API IVS com a AWS CloudTrail. | ✓ | ✓ | ✓ |
| CloudWatch | Um serviço da AWS para monitorar aplicações, responder a alterações de mudanças de performance, otimizar o uso de recursos e fornecer insights sobre integridade operacional. Você pode usar CloudWatch para monitorar as métricas do IVS; consulte Monitoramento do streaming em tempo real do IVS e Monitoramento do streaming de baixa latência do IVS. | ✓ | ✓ | ✓ |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Composição | O processo de combinar fluxos de áudio e vídeo de várias fontes em um único fluxo. | ✓ | ✓ | |
| Pipeline de composição | Uma sequência de etapas de processamento necessárias para combinar vários fluxos e codificar o fluxo resultante. | ✓ | ✓ | |
| Compactação | Codificação de informações usando menos bits do que a representação original. Qualquer compactação específica pode ser sem perdas ou com perdas. A compactação sem perdas reduz os bits ao identificar e eliminar a redundância estatística. Nenhuma informação é perdida na compactação sem perdas. A compactação com perdas reduz os bits ao remover informações desnecessárias ou menos importantes. | ✓ | ✓ | |
| Ambiente de gerenciamento | Armazena informações sobre os recursos do IVS, como canais, palcos ou salas de chat, e fornece interfaces para criar e gerenciar esses recursos. É regional (baseado nas regiões da AWS). | ✓ | ✓ | ✓ |
| CORS | O compartilhamento de recursos de origem cruzada é um recurso da AWS que permite que as aplicações Web clientes carregadas em um domínio interajam com recursos, como buckets do S3, em outro domínio. O acesso pode ser configurado com base em cabeçalhos, métodos HTTP e domínios de origem. Consulte Usar o compartilhamento de recursos de origem cruzada (CORS): Amazon Simple Storage Service no Guia do usuário do Amazon Simple Storage Service. | ✓ | | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Fonte de imagem personalizada | Uma interface fornecida pelo SDK de Transmissão do IVS que permite que uma aplicação forneça sua própria entrada de imagem em vez de ficar limitada a câmeras predefinidas. | ✓ | ✓ | |
| Plano de dados | A infraestrutura que transporta os dados da ingestão para a saída. Ele opera com base na configuração gerenciada no ambiente de gerenciamento e não está restrito a uma região da AWS. | ✓ | ✓ | ✓ |
| Codificador, codificação | O processo de conversão de conteúdo de vídeo e áudio em formato digital, adequado para streaming . A codificação pode ser baseada em hardware ou software. | ✓ | ✓ | |
| Evento | Uma notificação automática publicada pela IVS para o serviço de AmazonEventBridge monitoramento. Um evento representa uma alteração no estado ou na integridade de um recurso de streaming, como um palco ou um pipeline de composição. Consulte Usando a Amazon EventBridge com o IVS de baixa latência e Usando a Amazon EventBridge com o IVS Real-Time Streaming. | ✓ | ✓ | ✓ |
| FFmpeg | Um projeto de software gratuito e de código aberto que consiste em um conjunto de bibliotecas e programas para o tratamento de arquivos e fluxos de vídeo e áudio. O FFmpeg fornece uma solução entre plataformas para gravar, converter e transmitir áudio e vídeo. | ✓ | | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Fluxo fragmentado | Criado quando uma transmissão se desconecta e depois se reconecta no intervalo especificado na configuração de gravação do canal. Os vários fluxos resultantes são considerados uma única transmissão e são mesclados em um único fluxo gravado. Consulte Mesclar streams fragmentados em Gravação automática no Amazon S3 (streamin g de baixa latência). | ✓ | | |
| Taxa de quadros | Uma métrica de streaming para o número de quadros de vídeo transmitidos ou recebidos por segundo. | ✓ | ✓ | |
| HLS | HTTP Live Streaming (HLS), um protocolo de comunicações de streaming com taxa de bits adaptável baseado em HTTP usado para entregar fluxos do IVS aos espectadores. | ✓ | | |
| Lista de reprodução HLS | Uma lista dos segmentos de mídia que compõem um fluxo. As listas de reprodução HLS padrão são compostas de arquivos de mídia de dez segundos. O HLS também é compatível com listas de reprodução de intervalo de bytes mais granulare s. | ✓ | | |
| Host | O participante de um evento em tempo real que envia vídeo ou áudio para o palco. | | ✓ | |
| IAM | Identity and Access Management, um serviço da AWS que permite que os usuários gerenciem com segurança identidades e acesso aos serviços e recursos da AWS, incluindo o IVS. | ✓ | ✓ | ✓ |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Ingestão | Processo do IVS para receber fluxos de vídeo de um host ou transmissor para processamento ou entrega para visualizadores ou outros participantes. | ✓ | ✓ | |
| Servidor de ingestão | Recebe fluxos de vídeo e os envia para um sistema de transcodificação, em que os fluxos são submetidos a transmux ou são transcodificados para HLS para entrega aos visualizadores.<br><br>Os servidores de ingestão são componentes específicos do IVS que recebem fluxos para canais, junto com um protocolo de ingestão (RTMP, RTMPS). Consulte as informações sobre como criar um canal em Conceitos básicos do streaming de baixa latência do IVS. | | ✓ | |
| Vídeo entrelaçado | Transmite e exibe somente linhas pares ou ímpares de quadros subsequentes para criar uma duplicação percebida da taxa de quadros sem consumir largura de banda adicional. Não recomendamos o uso de vídeo entrelaçado devido a preocupações com a qualidade do vídeo. | ✓ | ✓ | |
| JSON | JavaScript Object Notation, um formato de arquivo de padrão aberto que usa texto legível por humanos para transmitir objetos de dados que consistem em pares de valores de atributos e tipos de dados de matriz ou outros valores serializáveis. | ✓ | ✓ | ✓ |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Quadro-chave, quadro delta, intervalo de quadros-chave | O quadro-chave (também conhecido como intracodificado ou i-frame) é um quadro completo da imagem em um vídeo. Os quadros subsequentes, os quadros delta (também chamados de quadros previstos ou p-frame), contêm apenas informações que foram alteradas. Os quadros-chave aparecerão várias vezes em um fluxo, dependendo do intervalo do quadro-chave definido no codificador. | ✓ | ✓ | |
| Lambda | Um serviço da AWS para executar código (denominado funções do Lambda) sem provisionar qualquer infraestrutura de servidor. As funções do Lambda podem ser executadas em resposta a eventos e solicitações de invocação ou com base em um cronograma. Por exemplo, o Chat do IVS usa funções do Lambda para permitir a revisão de mensagens em uma sala de chat. | ✓ | ✓ | ✓ |
| Latência, glass-to-glass latência | É um atraso na transferência de dados. O IVS define os intervalos de latência como:<br>• Baixa latência: menos de três segundos<br>• Latência em tempo real: menos de 300 ms<br><br>A lass-to-glass latência G se refere ao atraso desde o momento em que uma câmera captura uma transmissão ao vivo até o momento em que a transmissão aparece na tela do espectador. | ✓ | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Codificação em camadas com transmissão simultânea | Permite a codificação e publicação simultâneas de vários fluxos de vídeo com diferentes níveis de qualidade. Consulte Streaming adaptável: codificação em camadas com a transmissão simultânea em Otimizações de streaming em tempo real. | | ✓ | |
| Manipulador de revisão de mensagem | Permite que os clientes do Chat do IVS analisem e filtrem automaticamente as mensagens de chat do usuário antes que elas sejam enviadas para a sala de chat. Ele é habilitado pela associação de uma função do Lambda com uma sala de chat. Consulte Creating a Lambda Function em Chat Message Review Handler. | | | ✓ |
| Mesclador | Um recurso dos SDKs de Transmissão Móvel do IVS que usa várias fontes de áudio e vídeo e gera uma única saída. Ele oferece suporte ao gerenciamento de vídeo na tela e a elementos de áudio que representam fontes, como câmeras, microfones, capturas de tela e áudio e vídeo gerados pela aplicação. A saída pode então ser transmitida para o IVS. Consulte Configuração de uma sessão de transmissão para mixagem no SDK de Transmissão do IVS: Guia de mixagem (streaming de baixa latência). | ✓ | | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Streaming de vários hosts | Combina fluxos de vários hosts em um único fluxo. Isso pode ser feito usando a composição do cliente ou do servidor.<br><br>O streaming de vários hosts permite diversos cenários, como convidar visualizadores para um palco para perguntas e respostas, competições entre hosts, chat por vídeo e hosts conversando entre si na frente de um grande público. | | ✓ | |
| Lista de reprodução multivariante | Um índice de todos os fluxos variantes disponíveis para uma transmissão. | ✓ | | |
| OAC | Origin Access Control, um mecanismo para restringir o acesso a um bucket do S3, para que conteúdo como um stream gravado possa ser servido somente por meio CloudFrontde CDN. | ✓ | | |
| OBS | Open Broadcaster Software, software gratuito e de código aberto para gravação de vídeo e streaming ao vivo. O OBS oferece uma alternativa (ao SDK de Transmissão do IVS) para editoração eletrônica. Streamers mais sofisticados familiarizados com o OBS podem preferi-lo por seus recursos avançados de produção, como transições de cena, mixagem de áudio e gráficos de sobreposição. | ✓ | ✓ | |
| Participante | Um usuário em tempo real conectado ao palco como host ou visualizador. | | ✓ | |
| Token de participante | Autentica o participante de um evento em tempo real quando ele entra em um palco. Um token de participante também controla se um participante pode enviar vídeo para o palco. | | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Token de reprodução, par de chaves de reprodução | Um mecanismo de autorização que permite que os clientes restrinjam a reprodução de vídeo em canais privados. Os tokens de reprodução são gerados em um par de chaves de reprodução.<br><br>Um par de chaves de reprodução é o par de chaves públicas e privadas usado para assinar e validar o token de autorização do visualizador para reprodução. Consulte Criar ou importar uma chave de reprodução em Configuração de canais privados e consulte os endpoints do par de chaves de reprodução na IVS Low-Latency Streaming API Reference.. | ✓ | | |
| URL de reprodução | Identifica o endereço que um visualizador usa para iniciar a reprodução de um canal específico. E esse endereço pode ser usado em todo o mundo. O IVS automaticamente seleciona a melhor localização na rede de entrega de conteúdo global do IVS para entregar o vídeo a cada visualizador. Consulte as informações sobre como criar um canal em Conceitos básicos do streaming de baixa latência do IVS. | ✓ | | |
| Canal privado | Permite que os clientes restrinjam o acesso aos fluxos usando um mecanismo de autorização baseado em tokens de reprodução. Consulte Fluxo de trabalho para canais privados em Configuração de canais privados. | ✓ | | |
| Vídeo progressivo | Transmite e exibe todas as linhas de cada quadro em sequência. Recomendamos o uso de vídeo progressivo em todas as etapas de uma transmissão. | ✓ | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Cotas | Número máximo de recursos ou operações de serviço do IVS para sua conta da AWS. Ou seja, esses limites são para cada conta da AWS, salvo indicação em contrário. Todas as cotas são aplicadas por região. Consulte Amazon Interactive Video Service endpoints and quotas no Guia de referência geral da AWS. | ✓ | ✓ | ✓ |
| Regiões | Permitem acesso aos serviços da AWS que residam fisicamente em uma área geográfica específica. As regiões fornecem tolerância a falhas, estabilidade e resiliência e também podem reduzir a latência. Com as regiões, você pode criar recursos redundantes que permanecem disponíveis e não afetados por uma interrupção regional.<br><br>A maioria das solicitações de serviços da AWS está associada a uma região geográfica específica. Os recursos que você cria em uma região não existe em qualquer outra região, a menos que você use explicitamente um recurso de replicação oferecido por um serviço da AWS. Por exemplo, o Amazon S3 oferece suporte à replicação entre regiões. Alguns serviços, como o IAM, não têm recursos entre regiões. | ✓ | ✓ | ✓ |
| Resolução | Descreve o número de pixels em um único quadro de vídeo, por exemplo, Full HD ou 1080p define um quadro com 1920x1080 pixels. | ✓ | ✓ | |
| Usuário raiz | O proprietário de uma conta da AWS. O usuário raiz tem acesso completo a todos os serviços e recursos da AWS na conta da AWS. | ✓ | ✓ | ✓ |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| RTMP, RTMPS | O Real-Time Messaging Protocol é um padrão do setor para transmissão de vídeo em uma rede. O RTMPS é a versão segura do RTMP, sendo executado por uma conexão Transport Layer Security (TLS/SSL). | ✓ | ✓ | |
| Bucket do S3 | Uma coleção de objetos armazenados no Amazon S3. Muitas políticas, incluindo acesso e replicação, são definidas no bucket e se aplicam a todos os objetos no bucket. Por exemplo, uma transmissão do IVS é armazenada como vários objetos em um bucket do S3. | ✓ | | |
| SDK | Kit de desenvolvimento de software, uma coleção de bibliotecas para desenvolvedores que criam aplicações com o IVS. | ✓ | ✓ | ✓ |
| Segmentação de selfies | Permite substituir o plano de fundo em uma transmissão ao vivo, usando uma solução específica do cliente que aceita uma imagem da câmera como entrada e retorna uma máscara que fornece uma pontuação de confiança para cada pixel da imagem, indicando se ela está em primeiro ou segundo plano. Consulte Substituição de plano de fundo em SDK de Transmissão do IVS: filtros de câmera de terceiros (streaming em tempo real). | | ✓ | |
| Versionamento semântico | Um formato de versão na forma de Major.Minor.Patch. Correções de bugs que não afetam a API incrementam a versão do patch, as adições/alterações da API compatíveis com versões anteriores incrementam a versão secundária e as alterações da API incompatíveis com versões anteriores incrementam a versão principal. | ✓ | ✓ | ✓ |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Composição do servidor | Usa um servidor do IVS para mixar áudio e vídeo dos participantes do palco e, em seguida, enviar esse vídeo mixado para um canal do IVS para alcançar um público maior ou armazená-lo em um bucket do S3. A composição do servidor reduz a carga do cliente, melhora a resiliência da transmissão e permite um uso mais eficiente da largura de banda.<br><br>Consulte também a composição do cliente. | | ✓ | |
| Cotas de serviço | Serviço da AWS que ajuda a gerenciar as cotas para muitos serviços da AWS em um único local. Além de pesquisar os valores de cotas, você pode solicitar o aumento delas no console do Service Quotas. | ✓ | ✓ | ✓ |
| Perfil vinculado a serviço | Tipo exclusivo de perfil do IAM que é vinculado diretamente a um serviço da AWS. Os perfis vinculados a serviços são automaticamente criados pelo IVS e incluem todas as permissões que o serviço exige para chamar outros serviços da AWS em seu nome, por exemplo, acessar um bucket do S3. Consulte Uso de funções vinculadas ao serviço para o IVS em Segurança do IVS. | ✓ | | |
| Estágio | Recurso do IVS que representa um espaço virtual no qual os participantes do evento em tempo real podem trocar vídeos em tempo real. Consulte Criar um palco em Conceitos básicos do streaming em tempo real do IVS. | | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Sessão de palco | Começa quando o primeiro participante entra em um palco e termina alguns minutos após o último participante parar de publicar no palco. Um palco de longa duração pode ter várias sessões ao longo de sua vida útil. | | ✓ | |
| Fluxo | Dados que representam conteúdo de vídeo ou áudio que são enviados continuamente de uma origem para um destino. | ✓ | ✓ | |
| Chave de stream | Identificador atribuído pelo IVS quando você cria um canal. Usado para autorizar streaming para o canal. Trate a chave de fluxo como um segredo, pois qualquer pessoa que a tenha poderá fazer streaming para o canal. Consulte Conceitos básicos do streaming de baixa latência do IVS. | ✓ | | |
| Privação de fluxo | Um atraso ou uma interrupção na entrega de fluxo ao IVS. Isso ocorre quando o IVS não recebe a quantidade esperada de bits que o dispositivo de codificação anunciou que enviaria em um determinado período. Uma ocorrência de privação de fluxo resulta em um evento de privação de fluxo.<br><br>Do ponto de vista de um visualizador, a privação de fluxo pode ser um vídeo com atrasos, buffers ou congelamentos. A privação de fluxo pode ser breve (menos de cinco segundos) ou longa (vários minutos), dependendo da situação específica que resultou na privação de fluxo. Consulte O que é privação de fluxo? em Perguntas frequentes sobre solução de problemas. | ✓ | ✓ | |
| Streamer | Uma pessoa ou um dispositivo que envia um fluxo de vídeo ou áudio para o IVS. | ✓ | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Assinante | Participante de um evento em tempo real que recebe vídeo ou áudio dos hosts. Consulte O que é o Streaming em tempo real do IVS?. | | ✓ | |
| Tag | É um rótulo de metadados atribuído a um recurso da AWS. As tags ajudam a identificar e organizar os recursos da AWS. Na página de destino da documentação do IVS, consulte “Marcação” em qualquer documentação da API do IVS (para streaming em tempo real, streaming de baixa latência ou chat). | ✓ | ✓ | ✓ |
| Filtros de câmera de terceiros | Componentes de software que podem ser integrados ao SDK de Transmissão do IVS para permitir que um aplicação processe imagens antes de fornecê-las ao SDK de Transmissão como uma fonte de imagem personalizada. Um filtro de câmera de terceiros pode processar imagens da câmera, aplicar um efeito de filtro etc. | ✓ | ✓ | |
| Miniatura | Uma imagem de tamanho reduzido obtida de um fluxo. Por padrão, as miniaturas são geradas a cada 60 segundos, mas um intervalo menor pode ser configurado. A resolução da miniatura depende do tipo de canal. Consulte Conteúdo do registro em Gravação automática no Amazon S3 (streaming de baixa latência). | ✓ | | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Metadados temporizados | Metadados vinculados a carimbos de data e hora específicos em um fluxo. Eles podem ser adicionados programaticamente por meio da API do IVS e tornam-se associados a quadros específicos. Isso garante que todos os visualizadores recebam os metadados no mesmo ponto em relação ao fluxo.<br><br>Metadados sincronizados podem ser usados para acionar ações no cliente, como atualizar as estatísticas da equipe durante um evento esportivo. Consulte Como inserir metadados em um stream de vídeo. | ✓ | | |
| Transcodificação | Converte vídeo e áudio de um formato para outro. Um fluxo de entrada pode ser transcodificado para um formato diferente em várias taxas de bits e resoluções para oferecer suporte a uma variedade de dispositivos de reprodução e condições de rede. | ✓ | ✓ | |
| Transmux | Repetição simples de um empacotamento de um fluxo ingerido para o Amazon IVS, sem recodificação do fluxo de vídeo. “Transmux” é a abreviação de multiplexação de transcodificação, um processo que altera o formato de um arquivo de áudio ou vídeo, mantendo alguns ou todos os fluxos originais. Realizar transmux resulta na conversão para um formato de contêiner diferente sem alterar o conteúdo do arquivo. Diferente de transcodificação. | ✓ | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Fluxos variantes | Um conjunto de codificações da mesma transmissão em vários níveis de qualidade distintos. Cada fluxo variante é codificado como uma lista de reprodução HLS separada. Um índice dos fluxos variantes disponíveis é chamado de lista de reprodução multivariante.<br><br>Depois que o reprodutor do IVS recebe uma lista de reprodução multivariante do IVS, ele pode escolher entre os fluxos variantes durante a reprodução, mudando continuamente de um para outro à medida que as condições da rede mudam. | ✓ | | |
| VBR | Taxa de bits variável, um método de controle de taxa para codificadores que usa uma taxa de bits dinâmica que muda durante a reprodução, dependendo do nível de detalhe necessário. É altamente recomendável não usar a VBR devido a problemas de qualidade de vídeo. Em vez disso, use CBR. | ✓ | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| Exibição | Trata-se de uma sessão de exibição exclusiva fazendo download ou reproduzindo vídeo de forma ativa. As visualizações são a base para a cota de visualizações simultâneas.<br><br>Uma exibição se inicia quando uma sessão de visualização dá início à uma reprodução de vídeo. Uma exibição termina quando uma sessão de visualização interrompe a reprodução de vídeo. A reprodução é o único indicador de audiência; não são consideradas heurísticas de engajamento, como níveis de áudio, foco da guia do navegador e qualidade do vídeo. Ao contabilizar as visualizações, o Amazon IVS não considera a legitimidade dos visualizadores nem tenta desduplicar visualizações localizadas, como a execução de vários reprodutores de vídeo em uma única máquina. Consulte Outras cotas em Service Quotas (streaming de baixa latência). | ✓ | | |
| Visualizador | Uma pessoa que recebe um fluxo do IVS. | ✓ | | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| WebRTC | Comunicação em tempo real na Web, um projeto de código aberto que fornece comunicação em tempo real aos navegadores da Web e às aplicações móveis. Ele permite que a comunicação de áudio e vídeo funcione dentro de páginas da web, permitindo a peer-to-peer comunicação direta, eliminando a necessidade de instalar plug-ins ou baixar aplicativos nativos.<br><br>As tecnologias por trás do WebRTC são implementadas como um padrão aberto da Web e estão disponíveis como APIs JavaScript regulares em todos os principais navegadores ou como bibliotecas para clientes nativos, como Android e iOS. | ✓ | ✓ | |

| Prazo | Descrição | LL | RT | Bate-papo |
|---|---|---|---|---|
| CHICOTE | Protocolo de ingestão WebRTC-HTTP, um protocolo baseado em HTTP que permite a ingestão de conteúdo com base no WebRTC em serviços de streaming e/ou CDNs. O WHIP é um rascunho do IETF desenvolvido para padronizar a ingestão de WebRTC.<br><br>O WHIP permite a compatibilidade com softwares como o OBS, oferecendo uma alternativa (ao SDK de transmissão do IVS) para editoração eletrônica. Streamers mais sofisticados familiarizados com o OBS podem preferi-lo por seus recursos avançados de produção, como transições de cena, mixagem de áudio e gráficos de sobreposição<br><br>O WHIP também é benéfico em situações em que o uso do SDK de transmissão do IVS não é viável ou preferido. Por exemplo, em configurações que envolvem codificadores de hardware, o SDK de transmissão IVS pode não ser uma opção. No entanto, se o codificador suportar WHIP, você ainda poderá publicar diretamente do codificador no IVS.<br><br>Consulte OBS e WHIP Support. | | ✓ | |
| WSS | WebSocket Seguro, um protocolo para estabelecer por meio WebSockets de uma conexão TLS criptografada. Ele está sendo usado para se conectar aos endpoints do Chat do IVS. Consulte Step 4: Send and Receive Your First Message em Getting Started with IVS Chat. | | | ✓ |

# Histórico do documento (streaming em tempo real)

## Alterações no Guia do usuário do streaming em tempo real

| Alteração | Descrição | Data |
| --- | --- | --- |
| Suporte para OBS e WHIP | Adicionou uma nova página. Este documento explica como usar codificadores compatíve is com WHIP, como OBS, para publicar no streaming em tempo real do IVS. O WHIP (WebRTC-HTTP Ingestion Protocol) é um rascunho do IETF desenvolvido para padronizar a ingestão de WebRTC. | 6 de fevereiro de 2024 |
| SDK de transmissão: Android 1.14.1, iOS 1.14.1, Web 1.8.0 | Número da versão e links de artefatos atualizados para a nova versão, nos guias do SDK de real-time-streaming transmissão: Android, iOS e Web. Na página de aterrissa gem da documentação do Amazon IVS, atualização dos links de referência do Broadcast SDK para indicar a nova versão. Consulte também as Notas de release do Amazon IVS para essa versão.<br><br>Para o Guia Android, adicionamos um novo | 1 de fevereiro de 2024 |

| | | |
|---|---|---|
| | problema conhecido (tamanho do vídeo menor que 176x176).<br><br>Para o Web Guide, adicionam os um novo problema conhecido. A solução alternativa é restringir a resolução do vídeo a 720p ao invocar ou. `getUserMedia getDisplayMedia`<br><br>Em Otimizações de streaming em tempo real, atualizamos a configuração da codificação em camadas com o Simulcast; agora isso está desativado por padrão. | |
| SDK de transmissão: Android 1.13.4, iOS 1.13.4, Web 1.7.0 | Número da versão e links de artefatos atualizados para a nova versão, nos guias do SDK de real-time-streaming transmissão: Android, iOS e Web. Na página de aterrissagem da documentação do Amazon IVS, atualização dos links de referência do Broadcast SDK para indicar a nova versão. Consulte também as Notas de release do Amazon IVS para essa versão. | 3 de janeiro de 2024 |
| Glossário do IVS | Ampliou o glossário, abordando termos do IVS em tempo real, baixa latência e bate-papo. | 20 de dezembro de 2023 |

| | | |
|---|---|---|
| Stage Health: novas CloudWatch métricas | Renomeou a métrica PacketLoss (Stage) para DownloadPacketLoss (Stage) e lançou CloudWatch métricas adicionais para streaming em tempo real do IVS:<br>• DownloadPacketLoss (Palco, Participante)<br>• DroppedFrames (Palco, Participante)<br>• SubscribeBitrate (Palco, Participante,MediaType)<br><br>Consulte Monitoramento do streaming em tempo real do IVS. | 7 de dezembro de 2023 |
| Políticas gerenciadas do IAM | Foram adicionadas duas políticas gerenciadas, IVS ReadOnlyAccess e FullAccess IVS. Consulte:<br>• A nova seção sobre Managed Policies for Amazon IVS na página Security.<br>• Alterações na Etapa 3: configurar permissões do IAM em Conceitos básicos do streaming de baixa latência do IVS. | 5 de dezembro de 2023 |

| | | |
|---|---|---|
| SDK de Transmissão: Android 1.13.2, iOS 1.13.2 | Número da versão e links de artefatos atualizados para a nova versão, nos guias do SDK de real-time-streaming transmissão: Android e iOS.<br><br>Na página de aterrissagem da documentação do Amazon IVS, atualização dos links de referência do Broadcast SDK para indicar a nova versão.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão. | 4 de dezembro de 2023 |
| SDK de Transmissão: Android 1.13.1 | Número da versão e links de artefatos atualizados para a nova versão, no guia do SDK de real-time-streaming transmissão: Android.<br><br>Na página de aterrissagem da documentação do Amazon IVS, atualização dos links de referência do Broadcast SDK para indicar a nova versão.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão. | 21 de novembro de 2023 |
| Service Quotas | A “Resolução de publicação do participante” foi alterada de 1080p para 720p. | 18 de novembro de 2023 |

| | | |
|---|---|---|
| SDK de Transmissão: Android 1.13.0, iOS 1.13.0 | Número da versão e links de artefatos atualizados para a nova versão, nos guias do SDK de real-time-streaming transmissão: Android e iOS.<br><br>Na página de aterrissagem da documentação do Amazon IVS, atualização dos links de referência do Broadcast SDK para indicar a nova versão.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão.<br><br>Também fizemos várias atualizações nas Otimizaçõ es de streaming. Entre outros aspectos, agora o recurso “Streaming adaptável : codificação em camadas com a transmissão simultâne a” exige aceitação explícita e é suportado somente nas versões recentes do SDK. | 17 de novembro de 2023 |

| | | |
|---|---|---|
| Gravação composta | Foram feitas as seguintes alterações:<br>• Adição de uma página Gravação composta para esse novo recurso.<br>• Atualização da Introdução ao Streaming em tempo real do IVS com endpoints do S3 na política em “Configurar permissões do IAM”.<br>• Atualização do Service Quotas com cotas de taxa de chamadas para os novos endpoints. | 16 de novembro de 2023 |

| | | |
|---|---|---|
| Composição do servidor (SSC) | A composição do servidor do IVS permite que os clientes transfiram a composição e a transmissão de uma etapa do IVS para um serviço gerenciado pelo IVS. A transmissão de SSC e RTMP para um canal são invocadas por meio de endpoints do ambiente de gerenciamento do IVS na região de origem do palco. Consulte:<br>• Introdução: adicionamos endpoints SSC à política em "Configurar permissões do IAM".<br>• Usando a Amazon EventBridge com IVS — adicionamos novas métricas.<br>• Composição do servidor: esse novo documento inclui uma visão geral e instruções de configuração.<br>• Service Quotas: adicionam os novos limites de taxa de chamadas e outras cotas.<br><br>Consulte também:<br>• Alterações listadas abaixo nas Alterações na referência de API do Streaming em tempo real do IVS. | 16 de novembro de 2023 |

| | | |
|---|---|---|
| | • Alterações listadas em Histórico do documento (streaming de baixa latência). | |
| SDK de Transmissão do IVS | Na Visão geral do SDK de Transmissão, atualizamos os Requisitos da plataforma > Plataformas nativas para esclarecer quais versões do SDK são suportadas e adicionamos “Navegadores móveis (iOS e Android)”.<br><br>No Guia de Transmissão Web, adicionamos “Limitações da Web móvel”. | 9 de novembro de 2023 |
| SDK de Transmissão do IVS | Adição de uma nova página sobre Filtros de câmera de terceiros. | 9 de novembro de 2023 |
| Conceitos básicos do streaming em tempo real do IVS | Atualizamos os procedimentos em Configurar permissões do IAM. | 20 de outubro de 2023 |
| Monitoramento do streaming em tempo real | Em CloudWatch Métricas: IVS Real-Time Streaming , adicionamos valores de amostra para dimensões. | 17 de outubro de 2023 |
| SDK de transmissão: Guia da Web | Fizemos várias alterações em Monitorar estado Silenciado da mídia de participantes remotos. | 17 de outubro de 2023 |

| | | |
|---|---|---|
| SDK de transmissão: Web 1.6.0 | Número da versão e links de artefatos atualizados para a nova versão, no guia do SDK de real-time-streaming transmissão: Web.<br><br>A página inicial da documentação do Amazon IVS indica a versão atual de referências do Broadcast SDK.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão.<br><br>No Guia da Web, em "Recuperar um MediaStream de um dispositivo", também excluímos as duas `max` linhas; a melhor prática é especificar somente`ideal`.<br><br>Em Otimizações de streaming em tempo real, adicionamos uma nova seção, Otimização da taxa de bits de áudio e suporte a estéreo. | 16 de outubro de 2023 |
| Stage Health: novas CloudWatch métricas | CloudWatch Métricas lançadas para streaming em tempo real do IVS. Consulte Monitoramento do streaming em tempo real do IVS. | 12 de outubro de 2023 |

| | | |
|---|---|---|
| SDK de transmissão: Android 1.12.1 | Número da versão e links de artefatos atualizados para a nova versão, no guia do SDK de real-time-streaming transmissão: Android.<br>Também foi adicionada uma nova seção, Uso de microfones Bluetooth.<br><br>A página inicial da documentação do Amazon IVS indica a versão atual de referências do Broadcast SDK.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão. | 12 de outubro de 2023 |
| SDK de Transmissão: Web 1.5.2 | Número da versão e links de artefatos atualizados para a nova versão, no guia do SDK de real-time-streaming transmissão: Web.<br><br>A página inicial da documentação do Amazon IVS indica a versão atual de referências do Broadcast SDK.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão. | 14 de setembro de 2023 |
| Conceitos básicos do streaming em tempo real do IVS | Em Android > Instalar o SDK de transmissão, adicionada vinculação de dados. | 12 de setembro de 2023 |

| | | |
|---|---|---|
| Tratamento de erros do SDK de transmissão | Adição de seções de “Tratamento de erros” aos Guias do SDK de transmissão: Web, Android e iOS. | 12 de setembro de 2023 |
| Conceitos básicos do streaming em tempo real do IVS | Em Distribuir tokens de participantes, foi adicionada uma nota Importante sobre não desenvolver funcional idades com base no formato de token atual. | 1º de setembro de 2023 |
| Conceitos básicos do streaming em tempo real do IVS | Em Configurar permissões do IAM, o conjunto de permissões foi atualizado. | 31 de agosto de 2023 |
| SDK de Transmissão: Web 1.5.1, Android 1.12.0 e iOS 1.12.0 | Número da versão e links de artefatos atualizados para a nova versão, nos guias do SDK de real-time-streaming transmissão: Web, Android e iOS.<br><br>Na página de aterrissagem da documentação do Amazon IVS, atualização dos links de referência do Broadcast SDK para indicar a nova versão.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão. | 23 de agosto de 2023 |

| | | |
|---|---|---|
| Lançamento do streaming em tempo real | As principais alterações na documentação acompanham esta versão. Renomeamos a documentação anterior para Streaming de baixa latência do IVS e publicamos a nova documentação Streaming em tempo real do IVS. A página inicial da documentação do IVS agora tem seções separadas para o streaming em tempo real e o streaming de baixa latência. Cada seção tem um Guia do usuário e uma Referência de API próprios.<br><br>Para visualizar outras alterações na documentação, consulte Histórico do documento (streaming de baixa latência). | 7 de agosto de 2023 |
| SDK de Transmissão: Web 1.5.0, Android 1.11.0 e iOS 1.11.0 | O número de versão e os links de artefato foram atualizados para a nova versão nos guias do SDK de transmissão: Web, Android e iOS.<br><br>Na página de aterrissagem da documentação do Amazon IVS, atualização dos links de referência do Broadcast SDK para indicar a nova versão.<br><br>Consulte também as Notas de release do Amazon IVS para essa versão. | 7 de agosto de 2023 |

# Alterações na Referência de API do Streaming em tempo real do IVS

| Alterações de API | Descrição | Data |
|---|---|---|
| Gravação composta | Adicionamos 4 StorageConfiguration endpoints e 7 objetos (DestinationDetail,, S3 Recording Configuration, S3DetailDestinationConfiguration, S3,,). StorageConfiguration StorageConfiguration StorageConfigurationSummary<br><br>Modificamos 3 objetos (Composição, Destino DestinationConfiguration). Isso afeta a GetCompos ition resposta e a StartComposition solicitação e a resposta. | 16 de novembro de 2023 |
| Composição do servidor | Adicionamos 8 Composições e EncoderConfigurati on pontos finais e 11 objetos (ChannelDestinatio nConfiguration, Composição CompositionSummary , Destino, DestinationConfiguration, Destinati onSummary, EncoderConfiguration, EncoderCo nfigurationSummary, GridConfiguration, LayoutCon figuration, e Vídeo). | 16 de novembro de 2023 |
| Integridade do Stage: novos dados de participa nte | Foram adicionados seis campos ao objeto Participa nte: `browserName` , `browserVersion` , `ispName`, `osName`, `osVersion` e `sdkVersion` . Isso afeta a GetParticipant resposta. | 12 de outubro de 2023 |
| Token de participante | Foi adicionada uma nota Importante sobre não desenvolver funcionalidades com base no formato de token atual. | 1º de setembro de 2023 |
| Lançamento do streaming em tempo real do IVS | As principais alterações na documentação acompanham esta versão. Renomeamos a documentação anterior para Streaming de baixa latência do IVS e publicamos a nova documentação | 7 de agosto de 2023 |

| Alterações de API | Descrição | Data |
| --- | --- | --- |
| | Streaming em tempo real do IVS. A página inicial da documentação do IVS agora tem seções separadas para o streaming em tempo real e o streaming de baixa latência. Cada seção tem um Guia do usuário e uma Referência de API próprios.<br><br>A Referência de API do Streaming em tempo real do IVS faz parte da documentação do streaming em tempo real do IVS. Anteriormente, ela era intitulad a Referência da API de palco do IVS. Seu histórico anterior é descrito em Histórico do documento (streaming de baixa latência). | |

# Notas de release (streaming em tempo real)

## 6 de fevereiro de 2024

### Suporte para OBS e WHIP

O IVS pode ser usado com codificadores compatíveis com WHIP, como o OBS, para publicar no streaming em tempo real do IVS. O WHIP (WebRTC-HTTP Ingestion Protocol) é um rascunho do IETF desenvolvido para padronizar a ingestão de WebRTC. Veja a nova página sobre OBS e WHIP Support.

## 1 de fevereiro de 2024

### SDK de transmissão do Amazon IVS: Android 1.14.1, iOS 1.14.1, Web 1.8.0 (streaming em tempo real)

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de transmissão na Web 1.8.0 | Documentação de referência: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference<br>• A codificação em camadas com transmissão simultânea agora está desativada por padrão.<br>• Corrigido um problema em que uma instância do Stage não se desconectava completamente quando um Stage era excluído ou quando um participante era desconectado do servidor. O SDK agora emite um `STAGE_CONNECTION_STATE_CHANGED` evento com um estado de `DISCONNECTED` (em vez de `ERRORED` e depois`CONNECTING` ). |

| Plataforma | Downloads e alterações |
| --- | --- |
| | • Foi corrigido o problema em que a publicaçã o falhava ao atualizar a estratégia com faixas de áudio ou vídeo vazias. |
| SDK de transmissão para Android 1.14.1 | Documentação de referência: https://aws.github .io/ amazon-ivs-broadcast-docs /1.14.1/android<br>• A codificação em camadas com transmiss ão simultânea agora está desativada por padrão.<br>• Atualizado `libWebRTC` de M108 para M119.<br>• Várias falhas foram corrigidas para melhorar a estabilidade geral.<br>• Foi adicionado suporte para publicação em estéreo. Isso pode ser ativado por meio do `StageAudioConfiguration` objeto.<br>• Corrigido um bug que causava um feed preto dos participantes após entrarem em uma sessão.<br>• `libWebRTC` Referências internas atualizad as para evitar conflitos de símbolos quando outras `libWebRTC` versões são incluídas no mesmo aplicativo host. |

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de transmissão para iOS 1.14.1 | Baixe para streaming em tempo real: https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip<br><br>Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios<br><br>• A codificação em camadas com transmissão simultânea agora está desativada por padrão.<br>• Atualizado `libWebRTC` de M108 para M119.<br>• Várias falhas foram corrigidas para melhorar a estabilidade geral.<br>• Foi adicionado suporte para publicação em estéreo. Isso pode ser ativado por meio de `IVSLocalStageStreamAudioConfiguration`.<br>• Corrigiu uma falha ao ativar o modo somente áudio para outros participantes.<br>• TTV aprimorada e tamanho binário reduzido. |

## Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| arm64-v8a | 5.223 MB | 13,118 MB |
| armeabi-v7a | 4.524 MB | 9.134 MB |
| x86_64 | 5.418 MB | 13.955 MB |
| x86 | 5,61 MB | 14.369 MB |

### Tamanho do SDK de transmissão: iOS

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| arm64 | 3.350 MB | 7.790 MB |

## 3 de janeiro de 2024

### SDK de transmissão do Amazon IVS: Android 1.13.4, iOS 1.13.4, Web 1.7.0 (streaming em tempo real)

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de transmissão na Web 1.7.0 | Documentação de referência: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference<br>• Aprimorado time-to-video para assinantes que ingressam em etapas.<br>• A `minAudioBitrateKbps` propriedade foi removida (não foi usada).<br>• Recuperação aprimorada da rede durante interrupções ou alterações na Internet. |
| SDK de transmissão para Android 1.13.4 | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.4/android<br>• StageAudioConfiguration agora suporta a configuração de se o cancelamento de eco deve ser ativado. |
| SDK de transmissão para iOS 1.13.4 | Baixe para streaming em tempo real: https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip |

| Plataforma | Downloads e alterações |
| --- | --- |
| | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.4/ios<br>• No iOS, aprimoramos o mecanismo de áudio para gravação e reprodução com foco na estabilidade e na capacidade de recuperação. Isso aprimora o suporte para mudanças de rota durante o uso, melhora a recuperação da bateria em casos extremos e reduz a quantidade de bloqueio da rosca principal.<br>• Corrigido um problema em que o microfone podia permanecer ativo mesmo depois de ser desconectado de um palco, deixando o indicador de privacidade do iOS ligado. (O SDK não estava processando o áudio recebido no momento.) |

## Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| 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 |

## Tamanho do SDK de transmissão: iOS

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| arm64 | 3,45 MB | 7,84 MB |

# 7 de dezembro de 2023

## Novas CloudWatch métricas

Renomeamos a métrica PacketLoss (Estágio) para DownloadPacketLoss (Estágio). Também lançamos CloudWatch métricas adicionais para streaming em tempo real do IVS:

- DownloadPacketLoss (Palco, Participante)
- DroppedFrames (Palco, Participante)
- SubscribeBitrate (Palco, Participante,MediaType)

Consulte Monitoramento de streaming em tempo real do IVS.

# 4 de dezembro de 2023

## SDK de Transmissão do Amazon IVS: Android 1.13.2 e iOS 1.13.2 (streaming em tempo real)

| Plataforma | Downloads e alterações |
|---|---|
| Todos os dispositivos móveis (Android e iOS) | • A configuração de supressão de ruído está disponível para que os desenvolvedores ativem/desativem a publicação. |
| SDK de Transmissão 1.13.2 para Android | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/android<br>• Melhorado o tempo necessário para carregar o vídeo (TTV) ao entrar no primeiro palco em uma sessão. |
| SDK de Transmissão 1.13.2 para iOS | Baixe para streaming em tempo real: https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip<br>Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/ios |

| Plataforma | Downloads e alterações |
| --- | --- |
| | • Nenhuma alteração no SDK em tempo real. |

### Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| arm64-v8a | 5,177 MB | 13,01 MB |
| armeabi-v7a | 4,485 MB | 9,045 MB |
| x86_64 | 5,352 MB | 13,808 MB |
| x86 | 5,547 MB | 14,192 MB |

### Tamanho do SDK de transmissão: iOS

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| arm64 | 3,45 MB | 7,82 MB |

## 21 de novembro de 2023

### SDK de Transmissão do Amazon IVS: Android 1.13.1 (streaming em tempo real)

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de Transmissão 1.13.1 para Android | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.1/android<br>• Correção de um problema que causava uma falha ao sair, liberar e voltar rapidamente ao mesmo palco. |

### Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
|---|---|---|
| arm64-v8a | 5,177 MB | 13,102 MB |
| armeabi-v7a | 4,485 MB | 9,046 MB |
| x86_64 | 5,353 MB | 13,809 MB |
| x86 | 5,547 MB | 14,192 MB |

## 17 de novembro de 2023

### SDK de Transmissão do Amazon IVS: Android 1.13.0 e iOS 1.13.0 (streaming em tempo real)

| Plataforma | Downloads e alterações |
|---|---|
| Todos os dispositivos móveis (Android e iOS) | • Atualização de Otimizações de streaming . Entre outros aspectos, agora o recurso “Streaming adaptável: codificação em camadas com a transmissão simultâne a” exige aceitação explícita e é suportado somente nas versões recentes do SDK.<br>• Foi melhorada a estabilidade dos palcos ao reduzir ocorrências de falhas raras.<br>• Melhora do tempo necessário para carregar o vídeo (TTV) ao entrar em um palco.<br>• Melhora da experiência com dispositivos Bluetooth.<br>• Otimização do uso da CPU e da memória pelo SDK e redução do tamanho da bibliotec a. |

| Plataforma | Downloads e alterações |
| --- | --- |
| | • Inclusão da classe `StageAudioManager` , que pode ser usada para definir parâmetros de captura e reprodução de áudio, incluindo predefinições para comunicação de voz, reprodução de mídia e muito mais. Para saber mais, acesse a nova página SDK de Transmissão do IVS: modos de áudio móvel.<br>• Adição de uma nova função `requestQualityStats` para exibir eventos estruturados de qualidade com base nas estatísticas do WebRTC.<br>• Adição de uma nova função para atualizar a taxa de bits de áudio. Ela é definido em objetos `LocalStageStream` exatamente como a configuração de vídeo, mas por meio de um novo objeto de configuração de áudio. |

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de Transmissão 1.13.0 para Android | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.0/android<br>• Agora, todos os métodos na interface `StageRenderer` são opcionais.<br>• Adição de compatibilidade com a visualização baseada em `Surfaceview` para um melhor desempenho. Os métodos `getPreview` existentes em `Session` e `StageStream` continuam retornando uma subclasse de `TextureView`, mas isso poderá mudar em uma versão futura do SDK.<br>  • Se sua aplicação depender especificamente de `TextureView`, você poderá continuar sem alterações. Você também poderá alternar de `getPreview` para `getPreviewTextureView` a fim de se preparar para a eventual alteração do `getPreview` retornado por padrão.<br>  • Se sua aplicação não precisar especificamente de `TextureView`, recomendamos mudar para `getPreviewSurfaceView` a fim de reduzir o uso de CPU e de memória.<br>• Agora, o SDK implementa um novo tipo de pré-visualização chamado `ImagePreviewSurfaceTarget` que funciona com o objeto Android Surface fornecido pela aplicação. Não se trata de uma subclasse do Android View, que oferece maior flexibilidade.<br>• Correção do caso em que o retorno de chamada `onFrame` para o participante |

| Plataforma | Downloads e alterações |
| --- | --- |
| | remoto é chamado na hora errada com o tamanho errado.<br>• Agora, `SurfaceSource # getInputSurface` está anotado com `@Nullable`. Seu código deverá verificá-lo antes de usá-lo.<br>• Adição de `UserId` e `attributes` a `ParticipantInfo`. As propriedades `UserId` e `attributes` são incorporadas ao token e as aplicações podem recuperá-las por meio de `ParticipantInfo` sempre que um participante ingressar.<br>• Agora, a captura da câmera e a renderização de pré-visualização têm como padrão 720 x 1280 ou a resolução de publicação (o que for maior) em 15 fps. Você pode ajustar a resolução e/ou o fps usando `StageVideoConfiguration # setCameraCaptureQuality`.<br>• Agora, o `IllegalArgumentException` exibido ao definir as propriedades de configuração inclui o valor fornecido na mensagem de exceção. |

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de Transmissão 1.13.0 para iOS | Baixe para streaming em tempo real: https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip<br><br>Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.0/ios<br><br>• Correção do problema no qual o SDK não alterava a configuração do vídeo se a configuração do vídeo fosse atualizada antes da publicação.<br>• Incorporação da correção do Google para uma vulnerabilidade de segurança do LibVPX (CVE-2023-5217). (Observe que o SDK do Android não exigiu nenhuma alteração para esse problema.)<br>• Aplicações que usam outras bibliotecas que incluam `libWebRTC` não terão mais conflitos com o SDK de Transmissão do IVS.<br>• Agora, todos os métodos no protocolo `IVSStageRenderer` estão marcados com `@optional` .<br>• Agora, os microfones e câmeras retornados por nossos SDKs têm uma ordem de classificação garantida, conforme documentado nos próprios SDKs.<br>• Agora, várias câmeras podem ter um valor de `true` para sua propriedade `isDefault` , uma para cada posição, conforme determinado pelo sistema operacional.<br>• Adição de `IVSStageAudioManager` , que permite um controle preciso sobre o `AVAudioSession` subjacente a fim de |

| Plataforma | Downloads e alterações |
| --- | --- |
| | proporcionar uma maior variedade de casos de uso da funcionalidade de palco.<br>• adicionado a `.UserIdParticipantInfo` |

### Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| 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 |

### Tamanho do SDK de transmissão: iOS

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| arm64 | 3,45 MB | 7,84 MB |

## 16 de novembro de 2023

### Gravação composta

Esse novo recurso permite a gravação da visualização composta de um Palco do IVS em um bucket do Amazon S3. Para obter mais informações, consulte:

- Gravação composta: uma nova página.
- Introdução ao streaming em tempo real do IVS: adicionamos endpoints do S3 na política em “Configurar permissões do IAM”.
- Service Quotas: adicionamos cotas de taxa de chamadas para os novos endpoints.

- Referência da API de streaming em tempo real IVS — adicionamos 4 StorageConfiguration endpoints e 7 objetos (DestinationDetail,, S3 RecordingConfiguration, S3DetailDestinationConfiguration, S3,,). StorageConfiguration StorageConfiguration StorageConfigurationSummary Também modificamos 3 objetos (composição, destino DestinationConfiguration); isso afeta a GetComposition resposta e a StartComposition solicitação e a resposta.

# 16 de novembro de 2023

## Composição do servidor

A composição do servidor do IVS permite que os clientes transfiram a composição e a transmissão de uma etapa do IVS para um serviço gerenciado pelo IVS. A composição do servidor e a transmissão de RTMP para um canal são invocadas por meio de endpoints do ambiente de gerenciamento do IVS na região de origem do palco. Para obter mais informações, consulte:

- Introdução ao streaming em tempo real do IVS: adicionamos endpoints do SSC na política em “Configurar permissões do IAM”.
- Usando a Amazon EventBridge com o IVS Real-Time Streaming — adicionamos novas métricas.
- Composição do servidor: esse novo documento inclui uma visão geral e instruções de configuração.
- Service Quotas (streaming em tempo real): adicionamos novos limites de taxa de chamadas e outras cotas.
- Referência da API de streaming em tempo real — adicionamos 8 composições e EncoderConfiguration endpoints e 11 objetos (ChannelDestinationConfiguration, Composição CompositionSummary, Destino DestinationConfiguration, DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration, LayoutConfiguration, e Vídeo).

No Guia do usuário do streaming de baixa latência do IVS, consulte:

- Habilitar vários hosts em um fluxo do IVS: adicionamos “Transmitindo um palco: composição do cliente vs. composição do servidor” e atualizamos “4. Transmitir o palco”.

# 16 de outubro de 2023

## SDK de transmissão do Amazon IVS: Web 1.6.0 (streaming em tempo real)

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de transmissão da Web 1.6.0 | Documentação de referência: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference<br>• Melhor tempo de gravação de vídeo (TTV).<br>• Adicionada a configuração `maxAudioBitrate` , compatível com até 128 kbps de canais de áudio mono ou estéreo. |

# 12 de outubro de 2023

## Novas CloudWatch métricas e dados de participantes

Lançamos CloudWatch métricas para streaming em tempo real do IVS. Consulte Monitoramento de streaming em tempo real do IVS.

Também adicionamos seis campos ao objeto de API Participante: `browserName`, `browserVersion`, `ispName`, `osName`, `osVersion` e `sdkVersion`. Isso afeta a GetParticipant resposta. Consulte a IVS Real-Time Streaming API Reference.

# 12 de outubro de 2023

## SDK de transmissão do Amazon IVS: Android 1,12.1 (streaming em tempo real)

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de transmissão do Android 1.12.1 | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.1/android |

| Plataforma | Downloads e alterações |
| --- | --- |
| | • Corrigido um bug que causava um erro ao chamar `BroadcastSession.setListener` . |

### Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| 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 setembro de 2023

### SDK de Transmissão do Amazon IVS: Web 1.5.2 (Transmissão em tempo real)

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de Transmissão da Web 1.5.2 | Documentação de referência: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference<br>• Correção de um bug que impedia a republicação com `refreshStrategy` quando o estado publicado entra em um estado `ERRORED`. |

# 23 de agosto de 2023

## SDK de Transmissão do Amazon IVS: Web 1.5.1, Android 1.12.0 e iOS 1.12.0 (streaming em tempo real)

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de Transmissão da Web 1.5.1 | Documentação de referência: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference<br>• Corrigido um bug com tipos internos de Maybe em TypeScript 5.<br>• Foi adicionada melhor detecção para compatibilidade com o Simulcast.<br>• Foram corrigidas duas condições de corrida com `refreshStrategy` ao tentar publicar.<br>• Corrigida uma condição de corrida com `refreshStrategy` ao tentar atualizar os participantes para assinatura. |
| Todos os dispositivos móveis (Android e iOS) | • Correção de um problema raro no qual a ação de publicação nunca é concluída.<br>• Foi melhorada a estabilidade dos palcos ao reduzir ocorrências de falhas raras.<br>• A estabilidade das etapas foi aprimorada resolvendo os problemas de condição de corrida causados pela rápida entrada/saída.<br>• Um novo método `setOnFrameCallback` foi adicionado em `ImageDevice` . Isso permite a observação à medida que os quadros passam pelo dispositivo em si, dando uma visão da proporção das imagens mais recentes. Esse método também pode ser usado para detectar quando o primeiro |

| Plataforma | Downloads e alterações |
| --- | --- |
| | quadro é apresentado para um participante remoto em um estágio. |
| SDK de Transmissão do Android 1.12.0 | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.0/android<br>• O Android 9 passou a ser compatível.<br>• Melhor uso e desempenho da CPU. |
| SDK de Transmissão do iOS 1.12.0 | Baixe para streaming em tempo real: https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip<br>Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.0/ios<br>• Correção da assinatura de `IVSDeviceDiscovery.createAudioSourceWithName` para retornar `IVSCustomAudioSource` em vez de `IVSCustomImageSource`. |

## Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| 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 |

### Tamanho do SDK de transmissão: iOS

| Arquitetura | Tamanho compactado | Tamanho descompactado |
|---|---|---|
| arm64 | 5,06 MB | 10,92 MB |

## 7 de agosto de 2023

### SDK de Transmissão do Amazon IVS: Web 1.5.0, Android 1.11.0 e iOS 1.11.0

| Plataforma | Downloads e alterações |
|---|---|
| SDK de Transmissão da Web 1.5.0 | Documentação de referência: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference<br>• Transmissão simultânea adicionada: quando habilitado, esse recurso permite que o publicador envie camadas de vídeo de alta e de baixa qualidade. Os inscritos selecionam automaticamente a qualidade ideal com base nas condições da rede. Consulte Optimizing Media. |
| Todos os dispositivos móveis (Android e iOS) | Transmissão simultânea adicionada: quando habilitado, esse recurso permite que o publicador envie camadas de vídeo de alta e de baixa qualidade. Os inscritos selecionam automaticamente a qualidade ideal com base nas condições da rede. Consulte “Habilitação ou desabilitação da codificação em camadas com a transmissão simultânea” nos Guias do SDK de Transmissão para Android e iOS. |

| Plataforma | Downloads e alterações |
| --- | --- |
| SDK de Transmissão do Android 1.11.0 | Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.11.0/android<br>• Correção de um problema em que a criação de muitos palcos às vezes resultava em uma falha. (O número exato de palcos depende do dispositivo.) |
| SDK de Transmissão do iOS 1.11.0 | Faça download para obter o streaming em tempo real: https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip<br>Documentação de referência: https://aws.github.io/ amazon-ivs-broadcast-docs /1.11.0/ios<br>• Correção da inscrição em `IVSDeviceDiscovery.createAudioSourceWithName` para retornar `IVSCustomAudioSource` em vez de `IVSCustomImageSource`. |

## Tamanho do SDK de transmissão: Android

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| 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 |

### Tamanho do SDK de transmissão: iOS

| Arquitetura | Tamanho compactado | Tamanho descompactado |
| --- | --- | --- |
| arm64 | 5,030 MB | 10,810 MB |

# 7 de agosto de 2023

## Streaming em tempo real

O Streaming em tempo real do Amazon Interactive Video Service (IVS) possibilita que você forneça streams ao vivo com uma latência que pode ser inferior a 300 milissegundos do host ao espectador.

As principais alterações na documentação acompanham esta versão. Atualmente, a página inicial da documentação do IVS tem seções separadas para o streaming em tempo real e o streaming de baixa latência. Cada seção tem um Guia do usuário e uma Referência de API próprios. Para obter detalhes sobre a documentação, consulte o Histórico do documento (para alterações na documentação do streaming em tempo real e de baixa latência). Para o streaming em tempo real, comece com o Guia do usuário do streaming em tempo real do IVS e com a Referência de API do streaming em tempo real do IVS.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.