aws

リアルタイムストリーミングユーザーガイド

Amazon IVS

Amazon IVS: リアルタイムストリーミングユーザーガイド

Table of Contents

Amazon IVS リアルタイムストリーミングとは

Amazon インタラクティブビデオサービス (IVS) リアルタイムストリーミングでは、アプリケーションにリアルタイムのオーディオとビデオを追加するのに必要なものがすべて揃っています。

強度:

- リアルタイムレイテンシー — レイテンシーの影響を受けやすいユースケース向けのアプリケーションを構築し、視聴者が IVS リアルタイムストリーミングに接続してエンゲージメントを維持できるようにします。ホストから視聴者まで、300 ミリ秒未満のレイテンシーでライブストリームを配信します。
- 高い同時実行性 — IVS リアルタイムストリーミングにより、大規模なインタラクションの可能性が生まれます。最大 10,000 人の視聴者に対応、最大 12 人のホストがバーチャルステージに参加できます。
- モバイル最適化 — IVS リアルタイムストリーミングはモバイルユースケースに最適化されており、さまざまなデバイスやネットワーク能力に対応しています。Android と iOS 用の Amazon IVS Broadcast SDK を統合することで、ユーザーはホストまたは視聴者として参加し、モバイルデバイスで高品質のライブストリームを楽しむことができます。

ユースケース:

- ゲストスポット – ホストがゲストを「ステージ上」にプロモーションでき、視聴者をホストにしてリアルタイムの交流ができるアプリケーションを作成します。
- バーサス (VS) モード – ホスト間ではサイドバイサイドの対戦体験を生み出し、視聴者はリアルタイムでその対戦を視聴できるようにします。
- オーディオルーム – リスナーをゲストとして会話に招待し、オーディオルームでのエンゲージメントを深めます。
- ライブビデオオークション — オークションをインタラクティブなビデオイベントに変え、リアルタイムのレイテンシーにで盛り上がりや完全性を維持します。

製品に関するドキュメントに加えて、https://ivs.rocks/ を参照してください。これは、公開済みコンテンツ (デモ、コードサンプル、ブログ投稿) を閲覧し、コストを見積もり、ライブデモを通じて Amazon IVS を体験するための専用サイトです。

グローバルソリューション、リージョナルコントロール

ストリーミングと視聴はグローバル

Amazon IVS を使用して、世界中の視聴者にストリーミングできます。

- ストリーミングすると、Amazon IVS はユーザーの近くの場所で自動的に動画を取り込みます。
- 視聴者はライブストリームを世界中で視聴できます。

つまり、「データプレーン」はグローバルです。データプレーンとは、ストリーミング/取り込み、視聴を指します。

コントロールはリージョナル

Amazon IVS データプレーンはグローバルですが、「コントロールプレーン」はリージョンに基づきます。コントロールプレーンとは、Amazon IVS コンソール、API、リソース (ステージ) を指します。

つまり、Amazon IVS は「リージョナルな AWS サービス」といえます。各リージョンの Amazon IVS リソースは、他のリージョンの類似リソースから独立しています。たとえば、あるリージョンで作成したステージは、他のリージョンで作成したステージとは無関係です。

リソースを使用するときは (ステージ作成など)、リソースを作成するリージョンを指定する必要があります。その後、リソースを管理するときは、リソースを作成したリージョンと同じリージョンで管理する必要があります。

...を使用した場合	...によりリージョンを指定します。
Amazon IVS コンソール	ナビゲーションバー右上の [リージョンの選択] ドロップダウンを使用します。
Amazon IVS API	適切なサービスエンドポイントの使用 「Amazon IVS リアルタイムストリーミング API リファレンス」を参照してください。 (SDK で API にアクセスする場合は、SDK の `region` パラメータをセットアップします。AWS で構築するツールを参照してください。)
AWS CLI	次のいずれかを実行します:

...を使用した場合	...によりリージョンを指定します。
	• CLI コマンドに`--region <aws-region>` を追加します。 • ローカルの AWS 設定ファイルにリージョンを追加します。

ステージが作成されたリージョンにかかわらず、どこからでも Amazon IVS にストリーミングでき、視聴者はどこからでも視聴できます。

IVS リアルタイムストリーミングの開始

このドキュメントでは、Amazon IVS Real-Time Streaming をアプリに統合する手順を説明します。

トピック

序章

前提条件

リアルタイムストリーミングを初めて使用する前に、次のタスクをすべて完了してください。手順については、「IVS 低レイテンシーストリーミングの開始」を参照してください。

- AWS 無料アカウントを作成する
- ルートユーザーと管理ユーザーを設定する

その他のリファレンス

- 「IVS Web Broadcast SDK リファレンス」
- 「IVS Android ブロードキャスト SDK リファレンス」
- 「IVS iOS ウェブブロードキャスト SDK リファレンス」
- 「IVS リアルタイムストリーミング API リファレンス」

リアルタイムストリーミング用語

言葉	説明	
ステージ	参加者がリアルタイムでビデオを送受信できる仮想スペース。	
ホスト	ローカルのビデオをステージに送信する参加者。	
表示者	ホストのビデオを受け取る参加者。	
Participant	ホストまたはビューアーとしてステージに接続したユーザー。	
参加者トークン	ステージに参加する参加者を認証するトークン。	
ブロードキャスト SDK	参加者がビデオを送受信できるようにするクライアントライブラリ。	

手順の概要

1. the section called “IAM アクセス許可を設定する” — ユーザーに基本的なアクセス権限を付与する AWS Identity and Access Management (IAM) ポリシーを作成し、そのポリシーをユーザーに割り当てます。
2. ステージの作成 — 参加者がリアルタイムでビデオを交換できる仮想スペースを作成します。
3. 参加者トークンの配布 — 参加者にトークンを送信して、ステージに参加できるようにします。
4. IVS Broadcast SDK の統合— ブロードキャスト SDK をアプリに追加して、参加者がビデオを送受信できるようにします: the section called “Web”、the section called “Android”、the section called “iOS”。

5. ビデオの公開とサブスクライブ — ステージにビデオを送信し、他のホストからビデオを受信します: the section called “Web”、the section called “Android”、the section called “iOS”。

IAM アクセス許可を設定する

次に、基本的なアクセス許可のセット (Amazon IVS ステージの作成、参加者トークンの作成など) が可能になる AWS Identity and Access Management (IAM) ポリシーを作成し、ユーザーにこのポリシーを割り当てる必要があります。新しいユーザーの作成時にアクセス許可を追加するか、あるいは既存のユーザーにアクセス許可を追加することができます。両方の手順を以下に示します。

詳細 (IAM ユーザーとポリシーについて、ポリシーをユーザーにアタッチする方法、Amazon IVS を使用してユーザーのアクションを制限する方法など) については、以下を参照してください。

- IAM ユーザーガイドの IAM ユーザーの作成
- IAM に関する Amazon IVS セキュリティと「IVS のマネージドポリシー」の情報。
- 「Amazon IVS のセキュリテイ」にある IAM 情報

Amazon IVS 用の既存の AWS マネージドポリシーを使用するか、ユーザー、グループ、またはロールのセットに付与する権限をカスタマイズする新しいポリシーを作成できます。両方のアプローチを以下にて説明します。

IVS のアクセス許可には既存のポリシーを使用してください。

ほとんどの場合、Amazon IVS には AWS マネージドポリシーを使用することになります。これらについては、「IVS Security」の「IVS 用マネージドポリシー」セクションで詳しく説明されています。

- `IVSReadOnlyAccess` AWS マネージドポリシーを使用して、アプリケーション開発者がすべての IVS Get および List API エンドポイント (低レイテンシーとリアルタイムストリーミングの両方) にアクセスできるようにします。
- `IVSFullAccess` AWS マネージドポリシーを使用して、アプリケーション開発者がすべての IVS API エンドポイント (低レイテンシーとリアルタイムストリーミングの両方) にアクセスできるようにします。

オプション: Amazon IVS アクセス許可の新しいポリシーを作成する

以下の手順に従います。

1. AWS マネジメントコンソールにサインインして、IAM コンソールを開きます。https://console.aws.amazon.com/iam/
2. ナビゲーションペインで、[ポリシー]、[ポリシーの作成] の順に選択します。[アクセス許可の指定] ウィンドウが開きます。
3. [アクセス許可の指定] ウィンドウで、[JSON] タブをクリックし、次の IVS ポリシーをコピーして [ポリシーエディタ] テキスト領域に貼り付けます。(このポリシーにはすべての Amazon IVS アクションが含まれるわけではありません。エンドポイントのアクセス権限は必要に応じて追加/削除 (許可/拒否) できます。IVS エンドポイントの詳細については、「IVS リアルタイムストリーミング API リファレンス」を参照してください。)

```
{
   "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. [アクセス許可の指定] ウィンドウを開いたまま、[次へ] を選択します (ウィンドウの一番下までスクロールすると表示されます)。[レビューと作成] ウィンドウが開きます。
5. [レビューと作成] ウィンドウで、ポリシーにポリシー名を入力します。オプションで説明を追加します。ユーザーを作成する時に必要になるので、ポリシーの名前を書きとめておきます (下記を参照してください)。ページの最下部で、[Create policy] (ポリシーの作成) を選択します。
6. IAM コンソールウィンドウが表示され、新しいポリシーが作成されたことを確認するバナーが表示されます。

新しいユーザーを作成し、アクセス許可を付与する

IAM ユーザーアクセスキー

IAM アクセスキーは、アクセスキー ID とシークレットアクセスキーで構成されます。これは AWS へのプログラムによるリクエストの署名に使用されます。アクセスキーがない場合は、AWS マネジメントコンソールから作成できます。ベストプラクティスとして、ルートユーザーのアクセスキーは作成しないでください。

シークレットアクセスキーを表示またはダウンロードできるのは、アクセスキーを作成するときのみです。後で回復することはできません。ただ、アクセスキーはいつでも新しく作成できます。必要な IAM アクションを実行するためのアクセス許可が必要です。

アクセスキーは、常に安全に保管してください。(例え Amazon からの問い合わせであっても) 第三者と共有しないでください。詳細については、「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

手順

以下の手順に従います。

1. ナビゲーションペインで [ユーザー]、[ユーザーの作成] の順に選択します。[ユーザーの詳細を指定] ウィンドウが開きます。
2. [ユーザーの詳細を指定] ウィンドウで次の手順を実行します。
   a. [ユーザーの詳細] で、作成する新しいユーザーの名前を入力します。
   b. [AWS マネジメントコンソールへのユーザーアクセスを提供] を選択します。
   c. [コンソールパスワード] で、[自動生成パスワード] を選択します。
   d. [ユーザーは次回サインイン時に新しいパスワードを作成する必要があります] を選択します。
   e. [次へ] をクリックします。[アクセス許可の設定] ウィンドウが開きます。
3. [アクセス許可の設定] で、[ポリシーを直接アタッチする] を選択します。[アクセス許可ポリシー] ウィンドウが開きます。
4. 検索ボックスに、IVS ポリシー名 (AWS マネージドポリシーまたは以前に作成したカスタムポリシー) を入力します。見つかったら、チェックボックスをオンにして、ポリシーを選択します。
5. ページの最下部で、[次へ] を選択します。[レビューと作成] ウィンドウが開きます。
6. [レビューと作成] ウィンドウで、ユーザーのすべての詳細が正しいことを確認してから、ウィンドウ最下部にある [ユーザーの作成] を選択します。
7. [パスワードの取得] ウィンドウが開き、コンソールサインインの詳細が表示されます。後で参照できるように、この情報を保存しておきます。完了したら、[ユーザーリストに戻る] を選択します。

既存のユーザーへのアクセス許可を追加する

以下の手順に従います。

1. AWS マネジメントコンソールにサインインして、IAM コンソールを開きます。https://console.aws.amazon.com/iam/
2. ナビゲーションペインで、 [Users (ユーザー)] を選択し、更新する既存のユーザー名を選択します。(名前をクリックして選択します。選択ボックスはチェックしないでください。)
3. 概要ページの[アクセス許可] タブで、[アクセス許可の追加] を選択します。[アクセス許可の追加] ウィンドウが開きます。
4. [既存のポリシーを直接アタッチ] を選択します。[アクセス許可ポリシー] ウィンドウが開きます。
5. 検索ボックスに、IVS ポリシー名 (AWS マネージドポリシーまたは以前に作成したカスタムポリシー) を入力します。ポリシーが見つかったら、チェックボックスをオンにして、ポリシーを選択します。

6. ページの最下部で、[次へ] を選択します。[レビュー] ウィンドウが開きます。
7. [レビュー] ウィンドウの下部にある [アクセス許可の追加] を選択します。
8. [Summary] (概要) ページで、IVS ポリシーが追加されたことを確認します。

ステージの作成

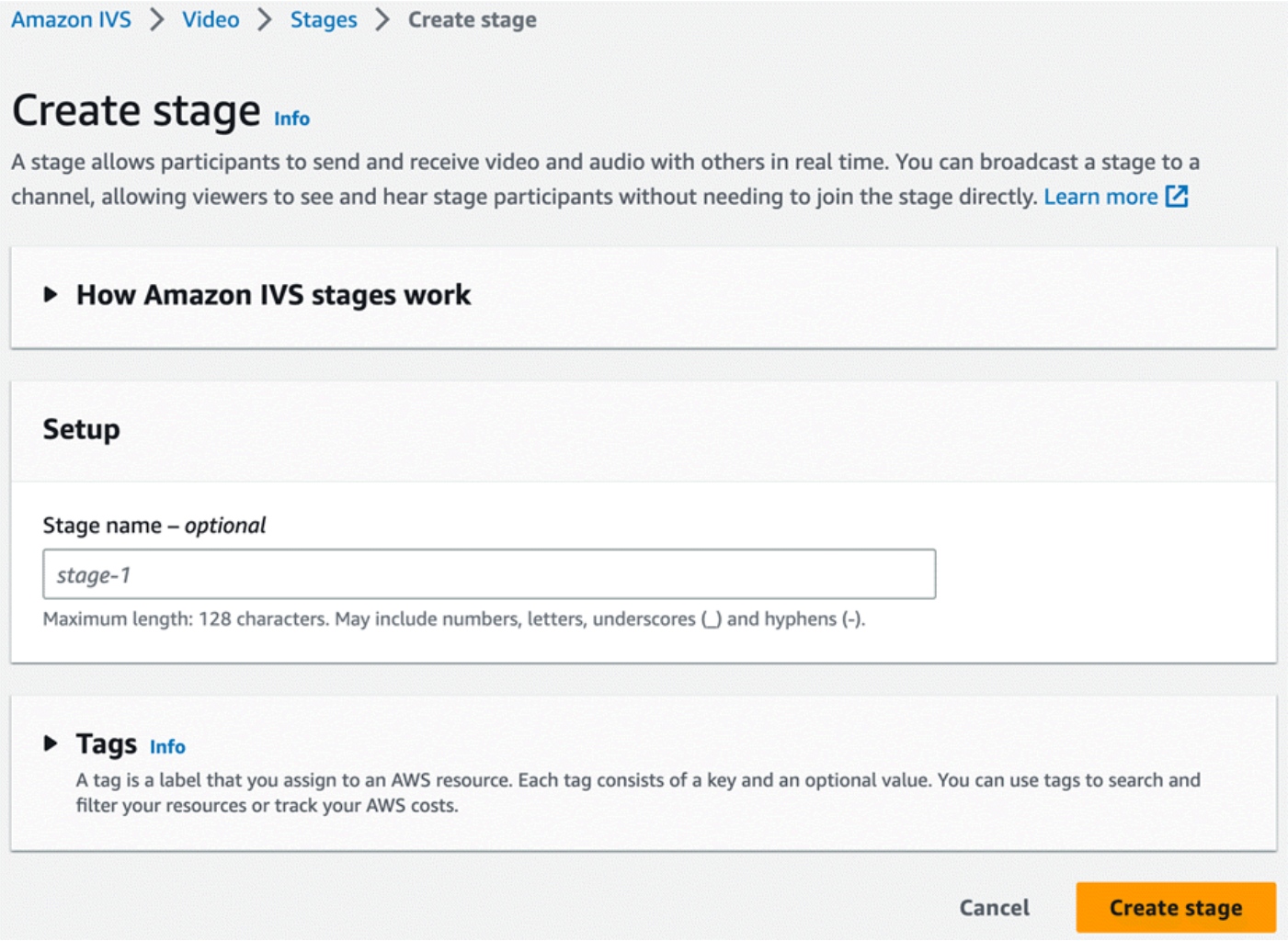
ステージは、参加者がリアルタイムでビデオを送受信できる仮想スペースです。リアルタイムストリーミング API の基盤となるリソースです。コンソールまたは CreateStage エンドポイントを使用してステージを作成できます。

可能な限り、再利用のために古いステージを保持するのではなく、論理セッ ションごとに新しいステージを作成し、終了したら削除することをお勧めします。古くなったリソース (再利用されない古いステージ) がクリーンアップされなければ、ステージの最大数の制限に早く到達する可能性が高くなります。

コンソールでの手順

1. Amazon IVS コンソールを開きます。

   (AWS マネジメントコンソールから Amazon IVS コンソールにアクセスすることもできます。)
2. 左側のナビゲーションペインで [ステージ] を選択し、[ステージを作成] を選択します。[ステージを作成] ウィンドウが表示されます。



3. 必要に応じて、[ステージ名] を入力します。[ステージを作成] を選択してステージを作成します。新しいステージのステージ詳細ページが表示されます。

CLI の手順

AWS CLI をインストールするには、「AWS CLI の最新バージョンをインストールまたは更新する」を参照してください。

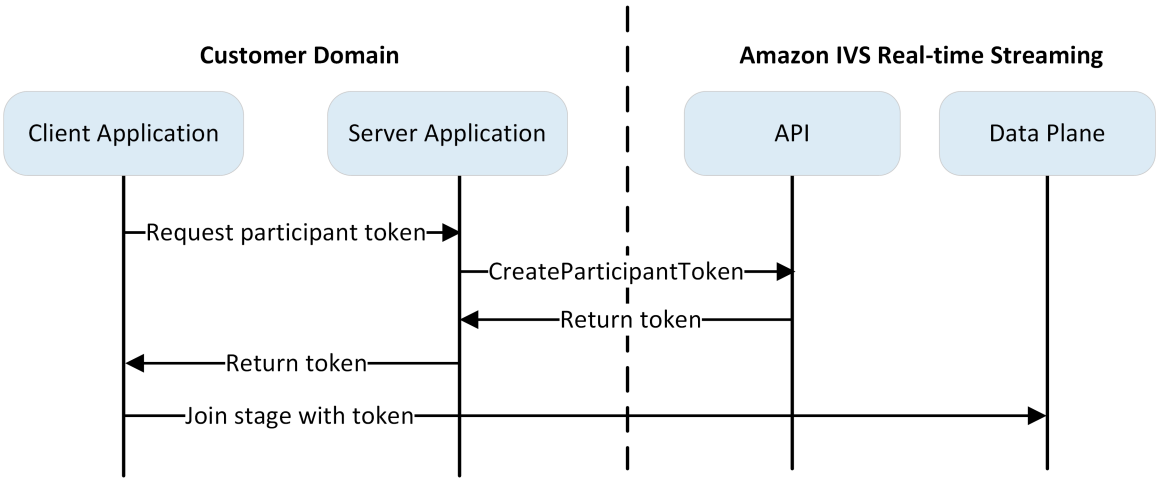
CLI を使用してリソースを作成し、管理できるようになりました。ステージ API は ivs-realtime 名前空間の下にあります。例えば、ステージを作成するには以下のようにします。

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

レスポンスは次のとおりです。

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

参加者トークンの配布



ステージができたら、トークンを作成して参加者に配布し、参加者がステージに参加してビデオの送受信を開始できるようにする必要があります。

上記のように、クライアントアプリケーションはサーバーアプリケーションにトークンを要求し、サーバーアプリケーションは AWS SDK または SigV4 署名付きリクエスト CreateParticipantToken を使用して を呼び出します。AWS 認証情報が API の呼び出しに使用されるため、トークンはクライアント側のアプリケーションではなく、安全なサーバー側のアプリケーションで生成する必要があります。

参加者トークンを作成するときに、オプションでそのトークンで有効にする機能を指定することができます。デフォルトは PUBLISH および SUBSCRIBE であり、これにより参加者はオーディオとビデオを送受信できるようになりますが、機能のサブセットを持つトークンを発行することもできます。たとえば、モデレーター向けに SUBSCRIBE 機能のみを備えたトークンを発行することができます。その場合、モデレーターはビデオを送信している参加者を見ることができますが、自分のビデオを送信することはできません。

テストや開発用に、コンソールまたは CLI を使用して参加者トークンを作成できますが、ほとんどの場合には、実稼働環境の AWS SDK を使用して作成することをお勧めします。

サーバーから各クライアントにトークンを配布する方法が必要になります (API リクエスト経由など)。この機能は提供していません。このガイドでは、以下の手順でトークンをコピーしてクライアントコードに貼り付けるだけです。

重要: トークンは不透明なものとして扱ってください。つまり、トークンの内容に基づいて機能を構築しないでください。トークンの形式は将来変更される可能性があります。

コンソールでの手順

1. 前のステップで作成したステージに移動します。
2. [参加者トークンを作成] を選択します。[参加者トークンの作成] ウィンドウが表示されます。
3. トークンに関連付けるユーザー ID を入力します。任意の UTF-8 でエンコードされたテキストを使用できます。
4. [参加者トークンを作成] を選択します。
5. トークンをコピーします。重要:トークンは必ず保存してください。IVS にトークンは保存されないため、後で取得することはできません。

CLI の手順

AWS CLI を使用してトークンを作成するには、最初に CLI をダウンロードし、コンピューターに設定する必要があります。詳細については、「AWS Command Line Interface のユーザーガイド」を参照してください。AWS CLI を使用してトークンを生成するのはテスト目的には適していますが、本番環境で使用する場合は、AWS SDK を使用してサーバー側でトークンを生成することをお勧めします (下記の手順を参照)。

1. ステージ ARN で `create-participant-token` コマンドを実行します。以下の機能のいずれか、またはすべてを含めます: `"PUBLISH"`、`"SUBSCRIBE"`。

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

2. これにより、参加者トークンが返されます。

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

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

3. このトークンを保存します。ステージに参加してビデオを送受信する際に必要となります。

AWS SDK での手順

AWS SDK を使用してトークンを作成することができます。以下は、 を使用した AWS SDK の手順です JavaScript。

重要: このコードは、サーバー側で実行し、その出力をクライアントに渡す必要があります。

前提条件： 以下のコードサンプルを使用するには、aws-sdk/client-ivs-realtime package をインストールする必要があります。詳細については、「AWS SDK for の開始方法 JavaScript」を参照してください。

```
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);
```

IVS Broadcast SDK の統合

IVS には、アプリケーションに統合できるウェブ、Android、iOS 用のブロードキャスト SDK が用意されています。ブロードキャストSDKは、ビデオの送信と受信の両方に使用されます。このセクションでは、2 人以上の参加者がリアルタイムで対話できるようにする簡単なアプリケーションを作成します。以下のステップでは、 という名前のアプリを作成する手順を説明します BasicRealTime。完全なアプリコードは CodePen と にあります GitHub。

- ウェブ: 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

ファイルのセットアップ

最初に、フォルダと最初の HTML および JS ファイルを作成してファイルを設定します。

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

ブロードキャスト SDK は、スクリプトタグまたは npm を使用してインストールできます。この例では、わかりやすくするためにスクリプトタグを使用していますが、後で npm を使用する場合でも簡単に変更できます。

スクリプトタグを使用する

Web Broadcast SDK は JavaScript ライブラリとして配布され、https://web-broadcast.live-video.net/1.8.0/amazon-ivs-web-broadcast.js で取得できます。

`<script>` タグを使用してロードすると、ライブラリは `IVSBroadcastClient` という名前のウィンドウスコープにグローバル変数を公開します。

npmを使う

npm パッケージをインストールします。

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

IVS BroadcastClient オブジェクトにアクセスできるようになりました。

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

Android

Android プロジェクトの作成

1. Android Studio で [新規プロジェクト] を作成します。
2. [空のビューアクティビティ] を選択します。

   注: Android Studio の一部の古いバージョンでは、ビューベースのアクティビティが [空のアクティビティ] になっていることがあります。お使いの Android Studio ウィンドウに [空のアクティビティ] と表示されており、[空のビュー] アクティビティが表示されない場合は、[空のアクティビティ] を選択してください。それ以外の場合は [空のアクティビティ] を選択しないでください。View API (Jetpack Compose ではなく) を使います。
3. プロジェクトに [名前] を付けて、[終了] を選択します。

ブロードキャスト SDK のインストール

Amazon IVS Android Broadcast ライブラリを Android 開発環境に追加するには、ここに説明されているように、ライブラリをモジュールの `build.gradle` ファイル (最新バージョンの Amazon IVS Broadcast SDK) に追加します。新しいプロジェクトの場合、`mavenCentral` リポジトリがすでに `settings.gradle` ファイルに含まれている場合があります。その場合は、`repositories` ブロックを省略することができます。このサンプルでは、`android` ブロック内でデータバインディングも有効にする必要があります。

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
}

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

または、SDK を手動でインストールするには、次の場所から最新バージョンをダウンロードします。

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

iOS

iOS プロジェクトの作成

1. 新しい Xcode プロジェクトを作成します。
2. [プラットフォーム] で iOS を選択します。
3. [アプリケーション] では [アプリ] を選択します。
4. アプリの [製品名] を入力し、[次へ] を選択します。
5. プロジェクトを保存するディレクトリを選択 (移動) し、[作成] を選択します。

次に、SDK を導入する必要があります。Broadcast SDK は 経由で統合することをお勧めします CocoaPods。代わりに、フレームワークを手動でプロジェクトに追加することもできます。両方の方法を以下に説明します。

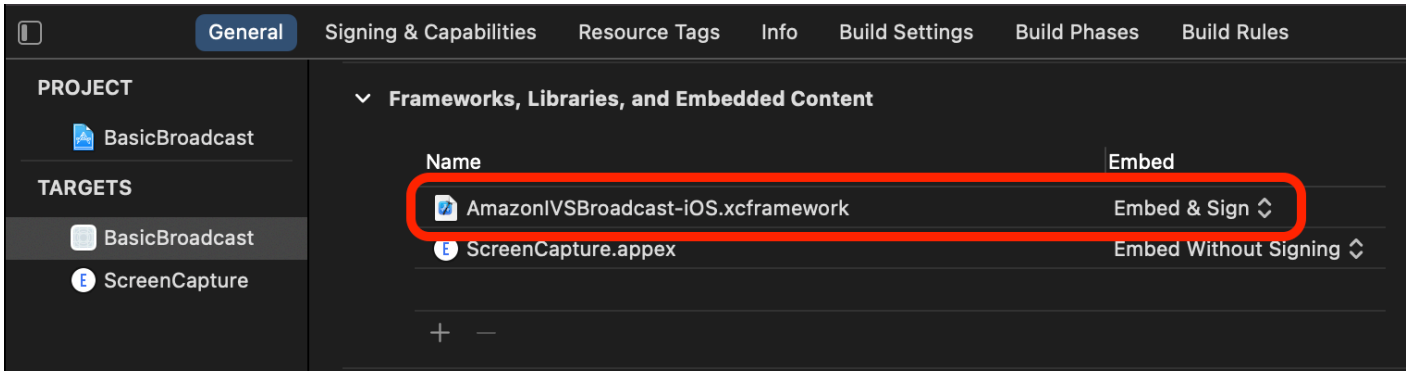
推奨: Broadcast SDK をインストールする (CocoaPods)

プロジェクト名が `BasicRealTime` だと仮定した場合、プロジェクトフォルダに次のコンテンツが含まれる `Podfile` を作成し、`pod install` を実行します。

```
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
```

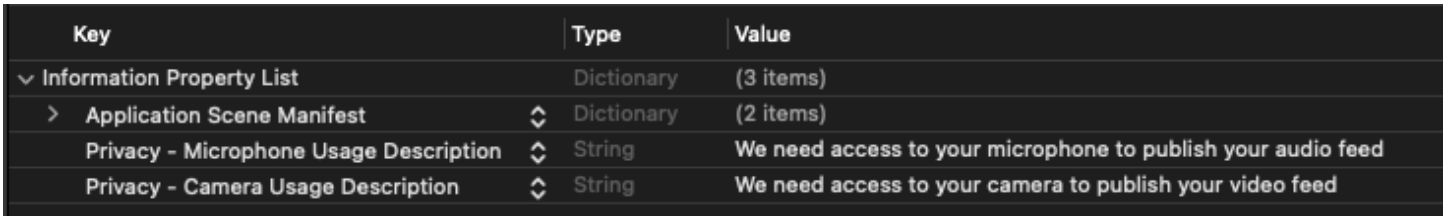
代替方法: フレームワークを手動でインストールする

1. https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip から最新バージョンをダウンロードします。
2. アーカイブの内容を抽出します。`AmazonIVSBroadcast.xcframework` には、デバイスとシミュレータの両方の SDK が含まれています。
3. アプリケーションターゲットの [全般] タブの、[Frameworks, Libraries, and Embedded Content (フレームワーク、ライブラリ、埋め込みコンテンツ)] のセクションに `AmazonIVSBroadcast.xcframework` をドラッグして埋め込みます。



のアクセス許可を設定します。

プロジェクトの Info.plist を更新して、NSCameraUsageDescription および NSMicrophoneUsageDescription に 2 つの新しいエントリを追加する必要があります。値には、アプリがカメラとマイクへのアクセスを要求する理由を説明する、ユーザー向けの説明を入力します。



ビデオの公開とサブスクライブ

ウェブ 、Android 、iOS については、以下の詳細を参照してください。 ???

Web

HTML 共通スクリプトの作成

最初に、HTML 共通スクリプトを作成し、ライブラリをスクリプトタグとしてインポートします。

```
<!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>
```

トークンの入力の受け入れと、参加/退出ボタンの追加

ここでは、Body に入力コントロールを入力します。これらはトークンを入力として受け取り、[参加] および [退出] ボタンを設定します。通常、アプリケーションはアプリケーションの API からトークンをリクエストしますが、この例では、トークンをコピーしてトークン入力に貼り付けます。

```
<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 />
```

メディアコンテナ要素の追加

これらの要素は、ローカル参加者およびリモート参加者向けのメディアを保持します。スクリプトタグを追加して、app.js で定義されているアプリケーションのロジックを読み込みます。

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

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

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

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

これで HTML ページが完成しました。ブラウザに `index.html` を読み込むとこれが表示されるはずです。

app.js の作成

次に、`app.js` ファイルのコンテンツを定義します。まず、SDK のグローバルから必要なすべてのプロパティをインポートします。

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

アプリケーション変数の作成

変数を構成し、[参加] および [退出] ボタンの HTML 要素への参照を保持し、アプリケーションの状態を保存します。

```
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;
```

joinStage 1 の作成: 関数の定義と入力の検証

joinStage 関数は入力トークンを受け取り、ステージへの接続を作成して、getUserMedia から取得したビデオとオーディオの公開を開始します。

まず関数を定義し、状態とトークンの入力を検証します。この機能については、この後のいくつかのセクションで説明します。

```
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
};
```

joinStage 2 の作成: メディアを公開する

次のメディアをステージに公開します。

```
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]);
```

joinStage 3 の作成: ステージ戦略の定義とステージの作成

このステージ戦略は、何を公開し、どの参加者にサブスクライブするかを決める際に SDK が使用する、決定ロジックの要となります。関数の目的の詳細については、「戦略」を参照してください。

この戦略はシンプルです。ステージに参加したら、直前に取得したストリームを公開し、リモート参加者全員のオーディオとビデオにサブスクライブします。

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

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

joinStage 4 の作成: ステージイベントの処理とメディアのレンダリング

ステージでは多くのイベントが発生します。STAGE_PARTICIPANT_STREAMS_ADDED および STAGE_PARTICIPANT_LEFT をリッスンして、ページ間でメディアをレンダリングしたり削除したりする必要があります。より詳細なイベントの一覧については、「イベント」にあるリストを参照してください。

ここでは、必要となる DOM 要素の管理に役立つヘルパー関数を 4 つ作成しています：setupParticipant、teardownParticipant、createVideoEl、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;
}
```

joinStage 5 の作成: ステージへの参加

ステージに参加して、joinStage 関数を完成させましょう。

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

leaveStage の作成

[退出] ボタンで呼び出す leaveStage 関数を定義します。

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

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

入力イベントハンドラーの初期化

最後にもう 1 つ、関数を app.js ファイルに追加します。この関数は、ページが読み込まれ、ステージへの参加と退出のためのイベントハンドラーが確立され次第すぐに呼び出されます。

```
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
```

アプリケーションの実行とトークンの提供

この時点でウェブページをローカルまたは他のユーザーと共有し、ページを開き、参加者トークンを入力してステージに参加できます。

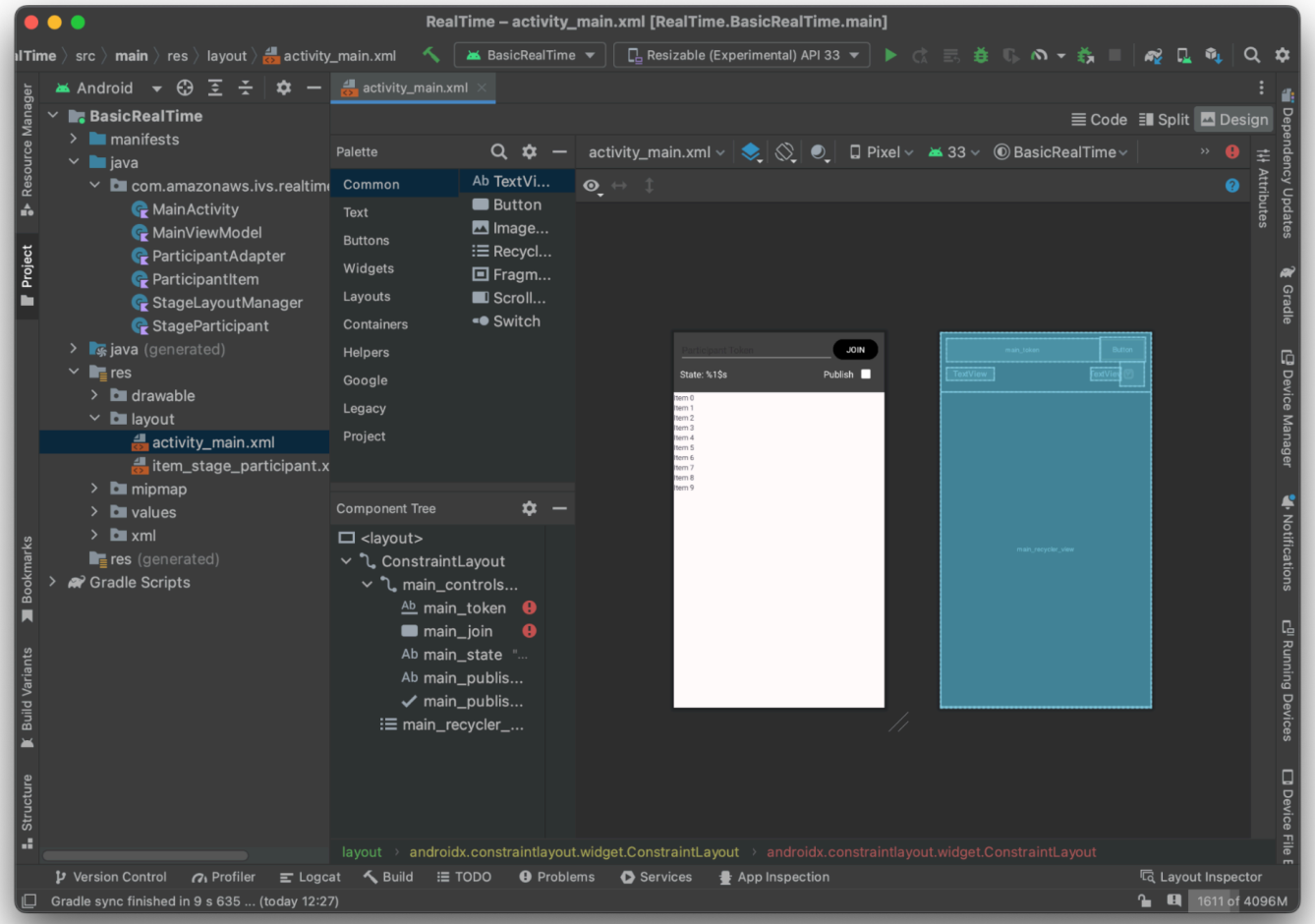
次のステップ

npm や React などに関する詳細な例については、「IVS Broadcast SDK: Web ガイド (リアルタイムストリーミングガイド」を参照してください。

Android

ビューの作成

まず、自動作成された `activity_main.xml` ファイルを使用して、アプリの簡単なレイアウトを作成します。レイアウトには、トークンを追加する `EditText`、参加 `Button`、ステージの状態を表示する `TextView`、公開/非公開を切り替える `CheckBox` が含まれます。



ビューの背景となる XML は次のとおりです。

```
<?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>
```

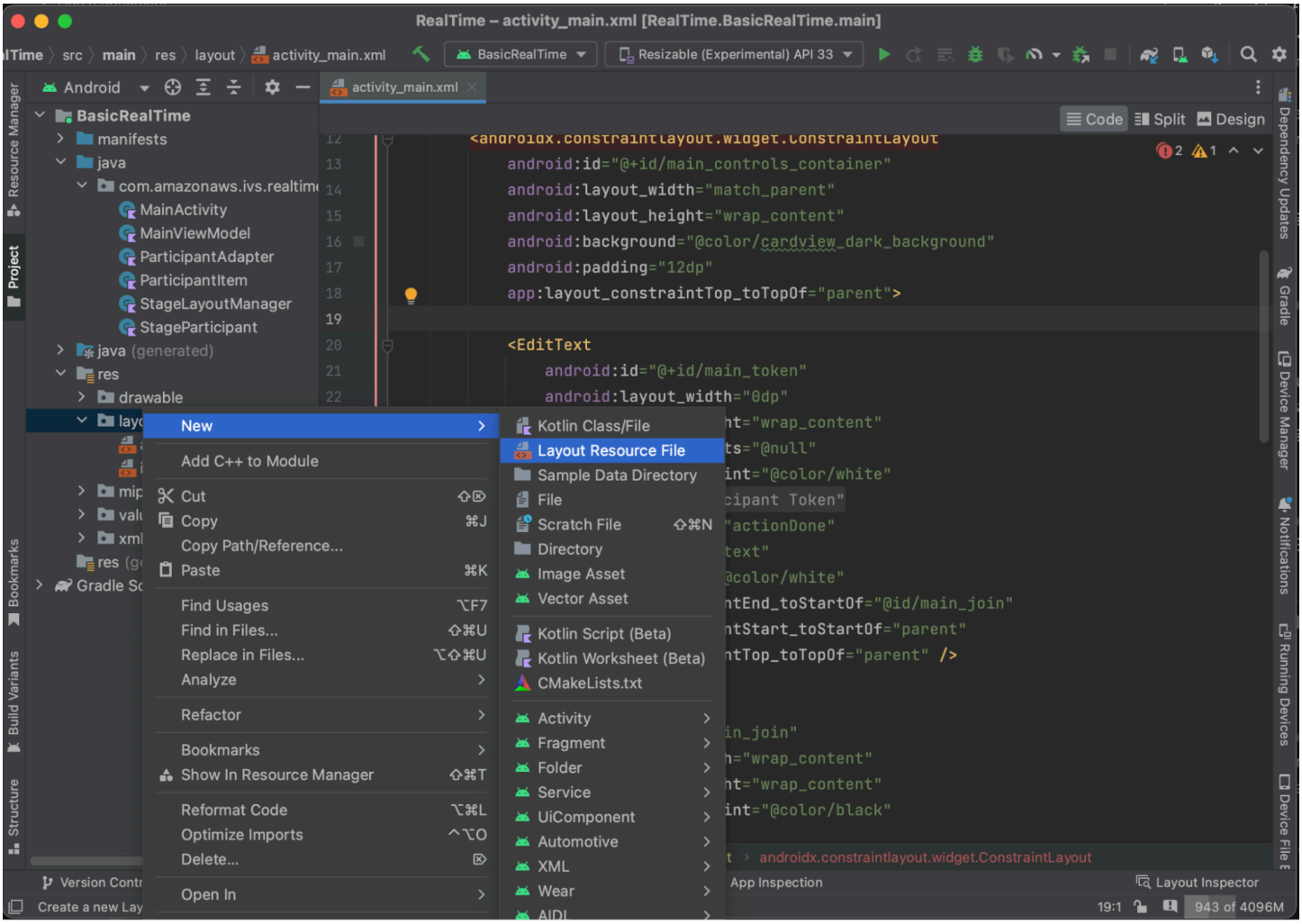
いくつかの文字列 ID を参照しました。全 `strings.xml` ファイルを作成しましょう。

```
<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>
```

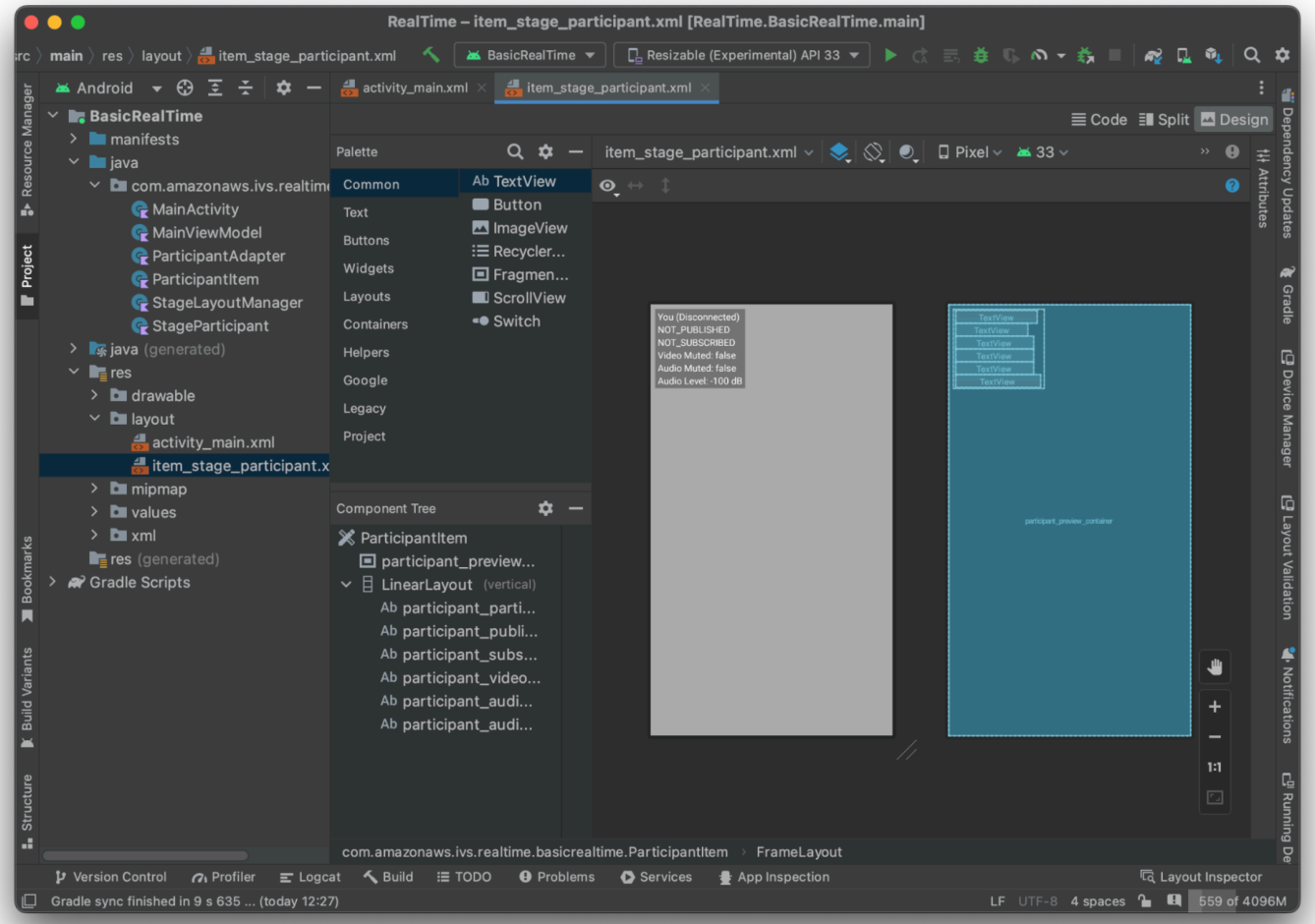
XML 内のこれらのビューを MainActivity.kt にリンクしましょう。

```
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)
}
```

次に、RecyclerView のアイテムビューを作成します。これを実行するには、res/layout ディレクトリを右クリックして [新規 > レイアウトリソースファイル] を選択します。この新しいファイルの名前は item_stage_participant.xml にします。



このアイテムのレイアウトはシンプルです。参加者のビデオストリームをレンダリングするためのビューと、参加者に関する情報を表示するためのラベルのリストが含まれています。



XML は次のようになります。

```
<?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>
```

この XML ファイルは、まだ作成していないクラス ParticipantItem を展開します。XML にはフル名前空間が含まれているため、この XML ファイルを必ず自分の名前空間に更新してください。このクラスを作成してビューを設定しましょう。または、とりあえず空白のままにしておくこともできます。

新しい Kotlin クラス ParticipantItem の作成:

```
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)
    }
}
```

アクセス許可

カメラとマイクを使用するには、ユーザーにアクセス許可をリクエストする必要があります。ここでは標準のアクセス許可フローに従います。

```
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
    }
}
```

アプリの状態

アプリケーションは参加者をローカルで に追跡MainViewModel.ktし、状態は Kotlin の MainActivityを使用して に返されますStateFlow。

新しい Kotlin クラス MainViewModel の作成:

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

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

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

}
```

MainActivity.kt で、ビューモデルを管理します。

```
import androidx.activity.viewModels

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

AndroidViewModel とこれらの Kotlin ViewModel 拡張機能を使用するには、モジュールの build.gradle ファイルに以下を追加する必要があります。

```
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 アダプター

簡単な RecyclerView.Adapter サブクラスを作成して参加者を追跡し、ステージイベント上の RecyclerView を更新します。まずはその前に、参加者を表すクラスが必要です。新しい Kotlin クラス StageParticipant の作成:

```
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)
            }
        }
}
```

このクラスは次に作成する ParticipantAdapter クラス内で使用します。まずクラスを定義し、参加者を追跡する変数を作成します。

```
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>()
```

また、残りのオーバーライドを実装する前に、RecyclerView.ViewHolder を定義する必要があります。

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

これを使用して、標準の RecyclerView.Adapter オーバーライドを実装できます。

```
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)
```

```
    }
}
```

最後に、参加者に変更が加えられたときに MainViewModel から呼び出す、新しいメソッドを追加します。これらのメソッドはアダプターの標準的な CRUD 操作です。

```
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])
    }
}
```

MainViewModel に戻り、このアダプターへの参照を作成して保持する必要があります。

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

ステージの状態

また、MainViewModel 内のステージ状態も追跡する必要があります。これらのプロパティを定義しましょう。

```
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>()
```

ステージに参加する前に自分のプレビューを確認できるように、ローカル参加者をすぐに作成します。

```
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)
}
```

これらのリソースは、ViewModel がクリーンアップされたときに確実にクリーンアップされるようにします。これらのリソースのクリーンアップを忘れないように、今すぐ onCleared() をオーバーライドします。

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

次に、権限が付与されたらすぐに、先ほど呼び出した permissionsGranted メソッドを実装してローカル streams プロパティに入力します。

```
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)
    }
}
```

ステージ SDK の実装

リアルタイム機能には、ステージ、ストラテジー、レンダラーという 3 つのコアコンセプトがあります。設計目標は、実際に動作する製品を構築するのに必要となるクライアント側ロジックの量を最小限に抑えることです。

Stage.Strategy

`Stage.Strategy` の実装は簡単です。

```
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
}
```

要約すると、内部の publishEnabled 状態に基づいて公開し、公開する場合には、以前に収集したストリームを公開します。このサンプルでは、常に他の参加者を購読して、オーディオとビデオの両方を受信しています。

StageRenderer

StageRenderer の実装も比較的簡単ですが、関数の数が多いことから含まれるコードの数がかなり多くなっています。このレンダラーの全体的なアプローチは、SDK から参加者の変更を通知されたときに ParticipantAdapter を更新するというものです。ローカル参加者が、参加する前にカメラのプレビューを確認できるように自分たちで管理することにしたため、一部のシナリオではローカル参加者の扱い方が異なる場合があります。

```
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) {}
}
```

カスタム の実装 RecyclerView LayoutManager

異なる人数の参加者をレイアウトするのは複雑です。親ビューのフレーム全体を占めるようにしたいものの、各参加者の設定を個別に処理するのは面倒です。これを簡単にするために、RecyclerView.LayoutManager の実装について順を追って説明します。

別の新しいクラスとなる StageLayoutManager を作成します。これが、GridLayoutManager を拡張します。このクラスは、フローベースの行/列レイアウトの参加者数に基づいて、各参加者のレイアウトを計算するように設計されています。各行の高さは他の行と同じですが、列の幅は行ごとに異なる場合があります。この動作をカスタマイズする方法については、layouts 変数の上のあるコードコメントを参照してください。

```
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
}
```

MainActivity.kt に戻り、RecyclerView のアダプターとレイアウトマネージャーを設定する必要があります。

```
// In onCreate after setting recyclerView.
recyclerView.layoutManager = StageLayoutManager(this)
recyclerView.adapter = viewModel.participantAdapter
```

UI アクションの接続

もうすぐ完了です。接続する必要がある UI アクションがあと少しあるだけです。

まず、MainActivity に MainViewModel からの StateFlow の変更を観察させます。

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

次に、[参加] ボタンと [公開] チェックボックスにリスナーを追加します。

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

上記の両方とも、MainViewModel の関数 (今から実装します) を呼び出します。

```
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
}
```

参加者のレンダリング

最後に、SDK から受け取ったデータを、先ほど作成した参加者アイテムにレンダリングする必要があります。RecyclerView ロジックはすでに完了しているので、ParticipantItem の bind API を実装するだけです。

空の関数を追加することから始めて、1 つずつ見ていきましょう。

```
fun bind(participant: StageParticipant) {

}
```

まず、簡易状態、参加者 ID、公開状態、サブスクライブ状態を処理します。これらについては、TextViews を直接更新します。

```
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
```

次に、オーディオとビデオのミュート状態を更新します。ミュート状態を取得するには、ストリーム配列から ImageDevice と AudioDevice を見つける必要があります。パフォーマンスを最適化するために、最後にアタッチされたデバイス ID を記憶しています。

```
// 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"
}
```

最後に、imageDevice のプレビューをレンダリングします。

```
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
```

そして、audioDevice からのオーディオ状態を表示します。

```
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
```

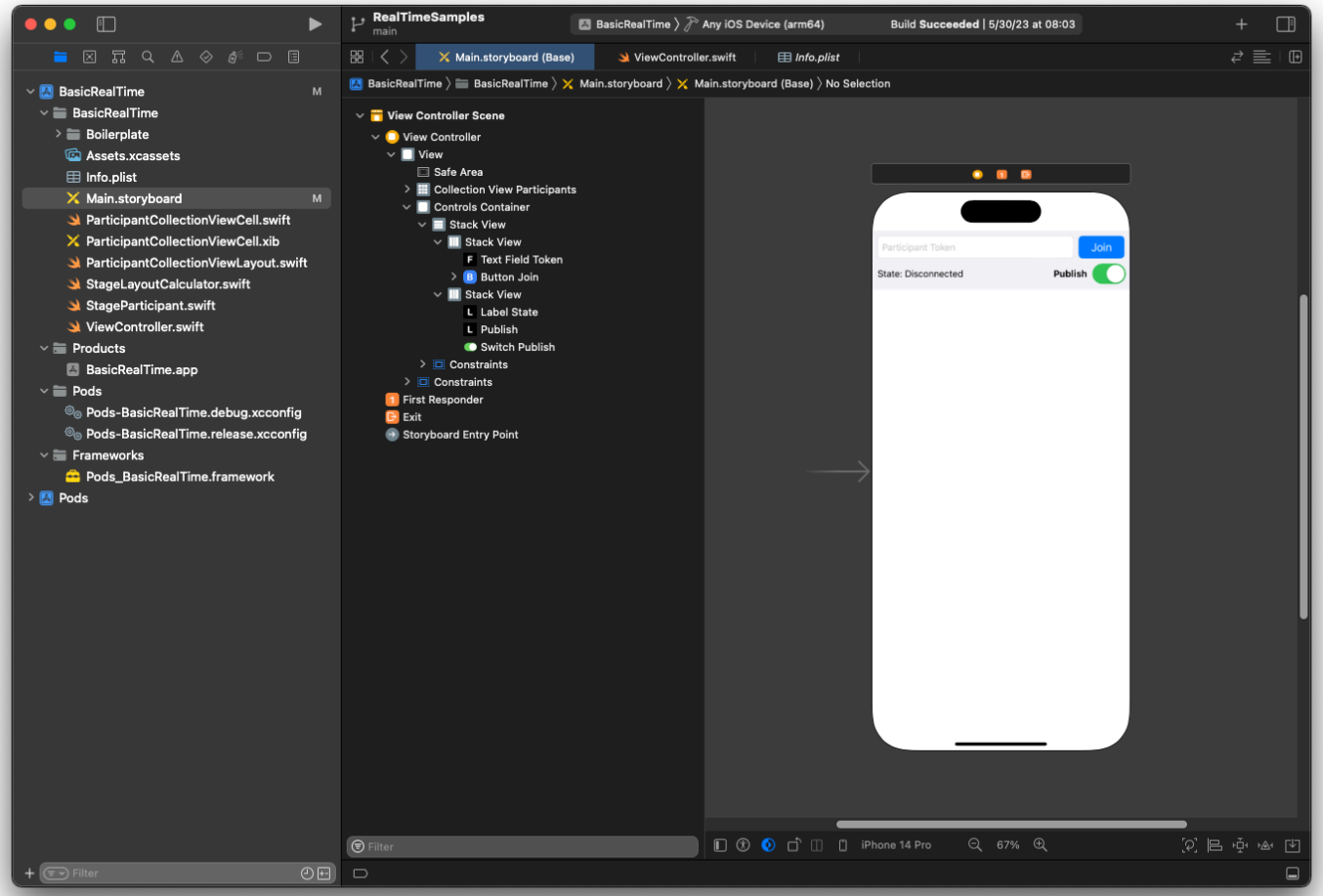
iOS

ビューの作成

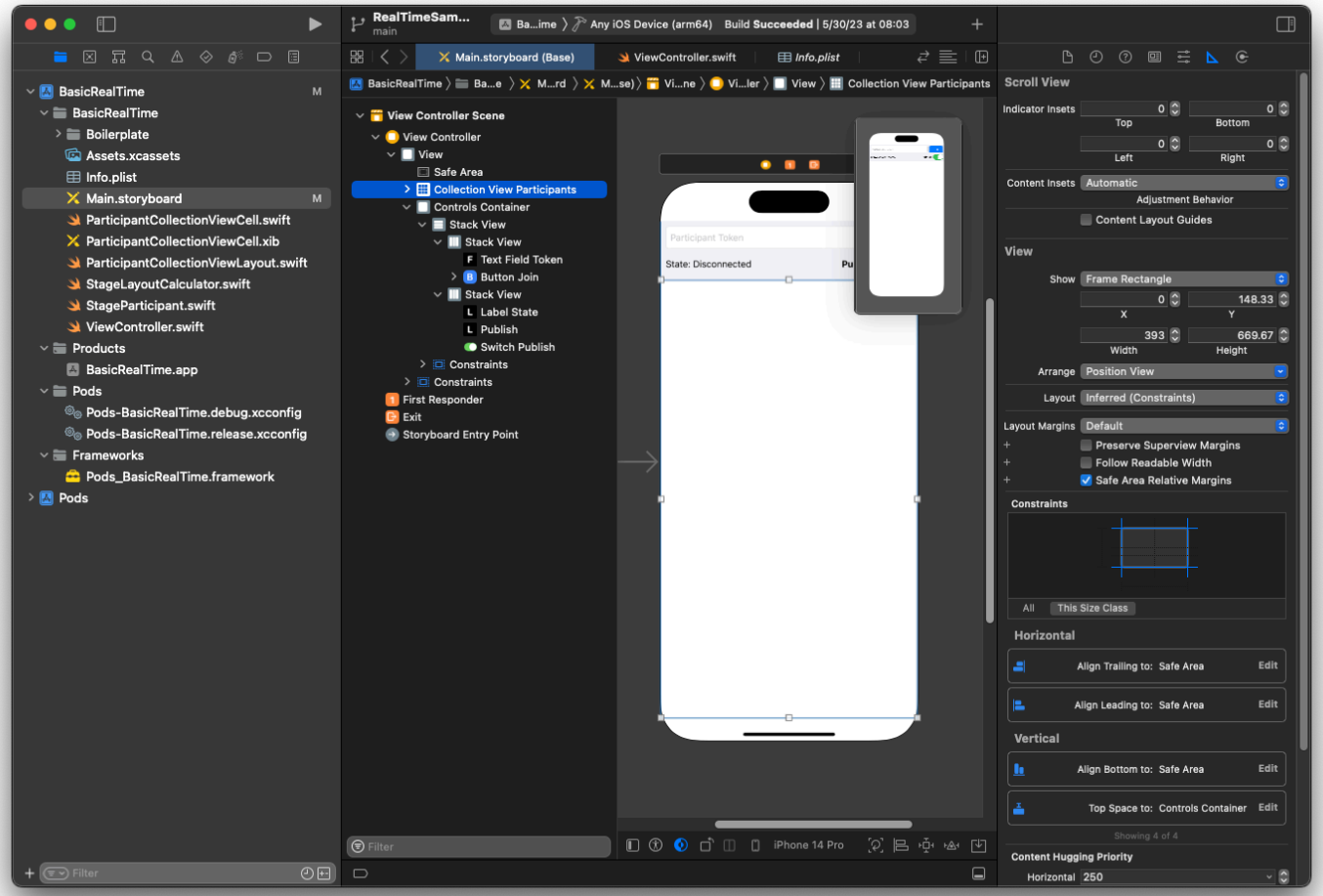
まず、自動作成された ViewController.swift ファイルを使用して AmazonIVSBroadcast をインポートし、リンクにいくつか @IBOutlets を追加します。

```
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!
```

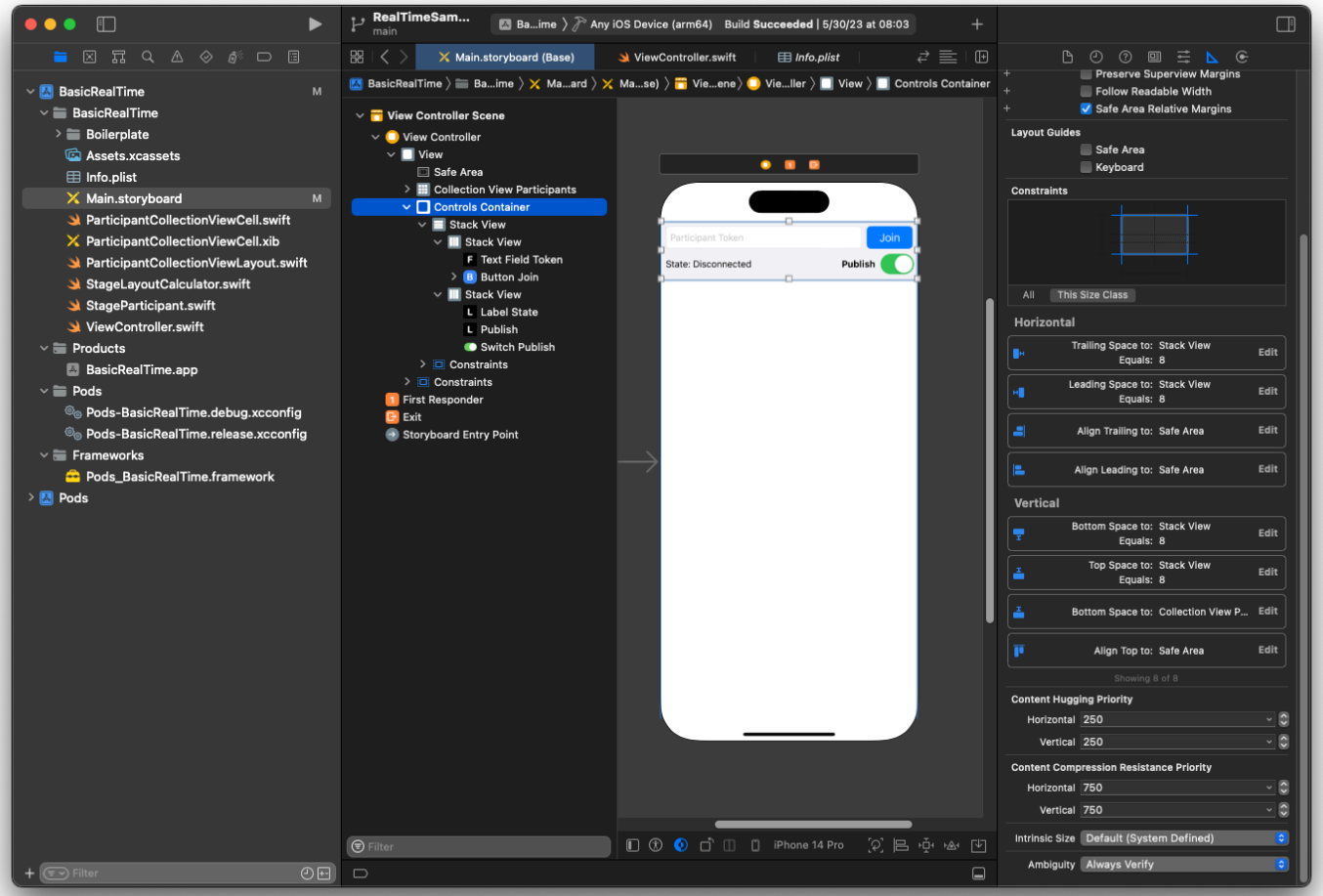
次に、これらのビューを作成して Main.storyboard でリンクします。使用するビュー構造は次のとおりです。



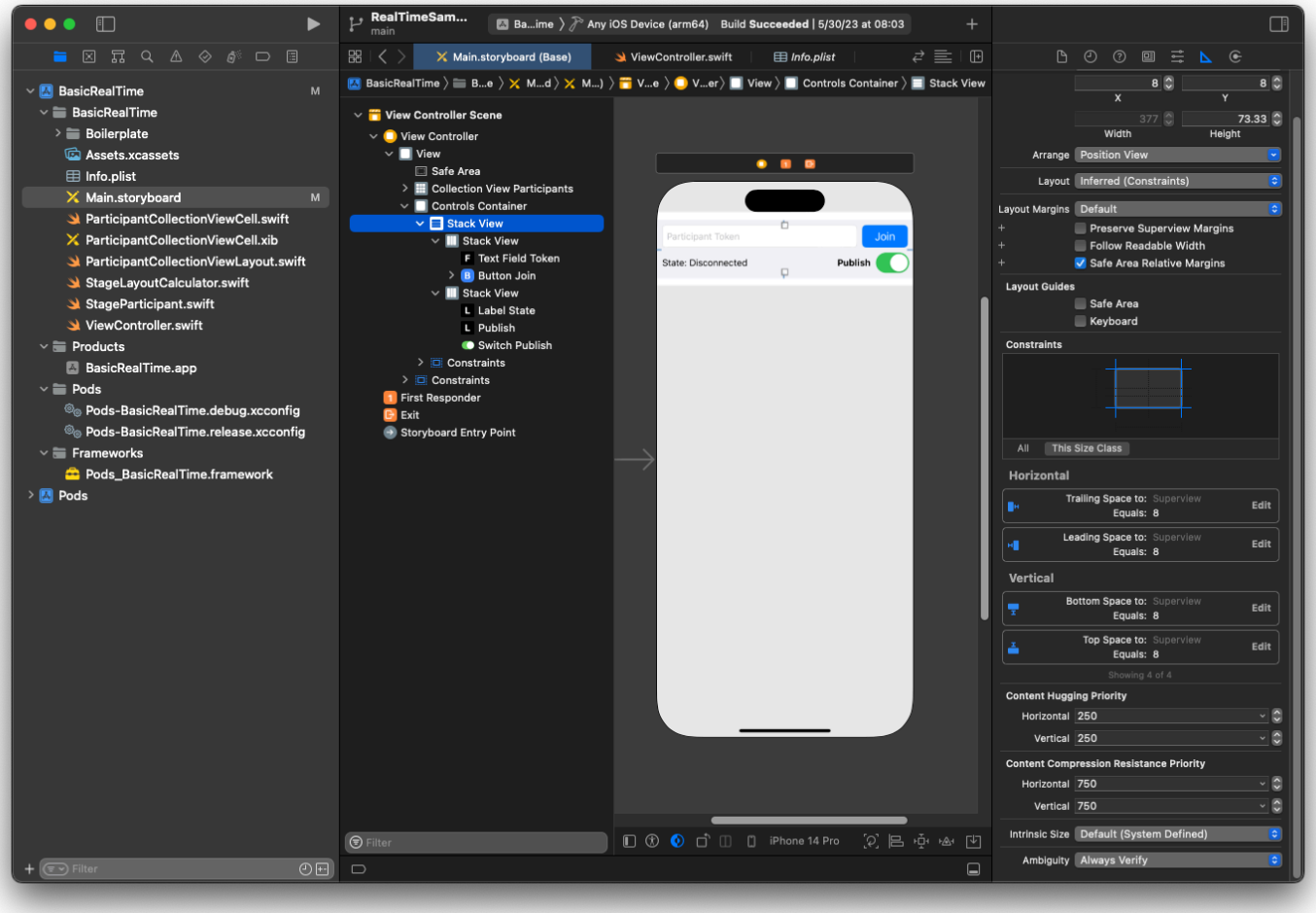
AutoLayout 設定では、3 つのビューをカスタマイズする必要があります。ズする必要があります。最初のビューは [Collection View Participants] (`UICollectionView`) です。[Leading]、[Trailing]、[Bottom] を [Safe Area] にバインドします。また、[Top] も [Controls Container] にバインドします。



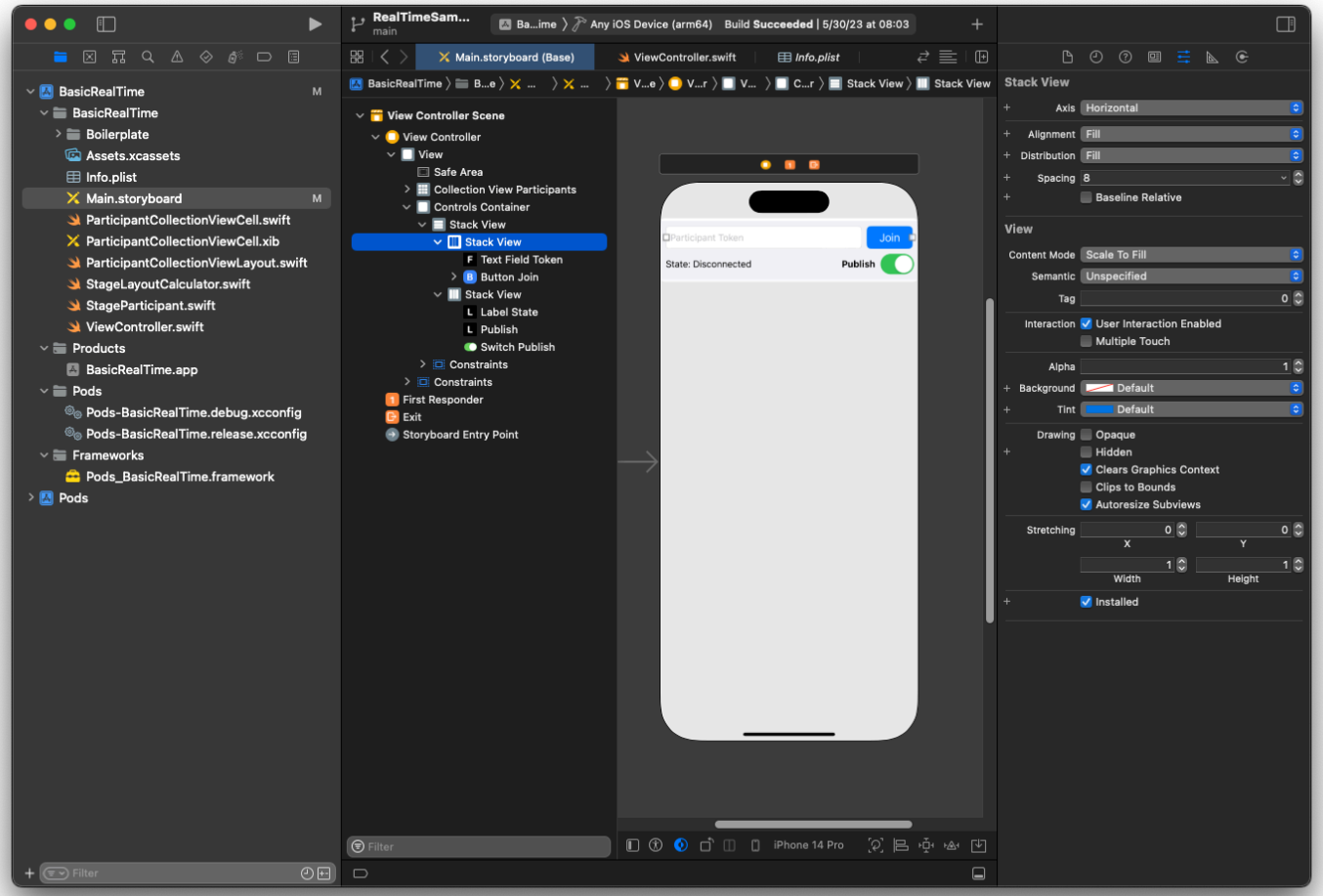
2 番目のビューは [コントロールコンテナ] です。[先頭]、[末尾]、[上] を [安全エリア] にバインドしました。



3 番目の最後のビューは [垂直スタックビュー] です。[上]、[先頭]、[末尾]、[下] を [Superview] にバインドしました。スタイルを設定するには、間隔を 0 ではなく 8 に設定します。



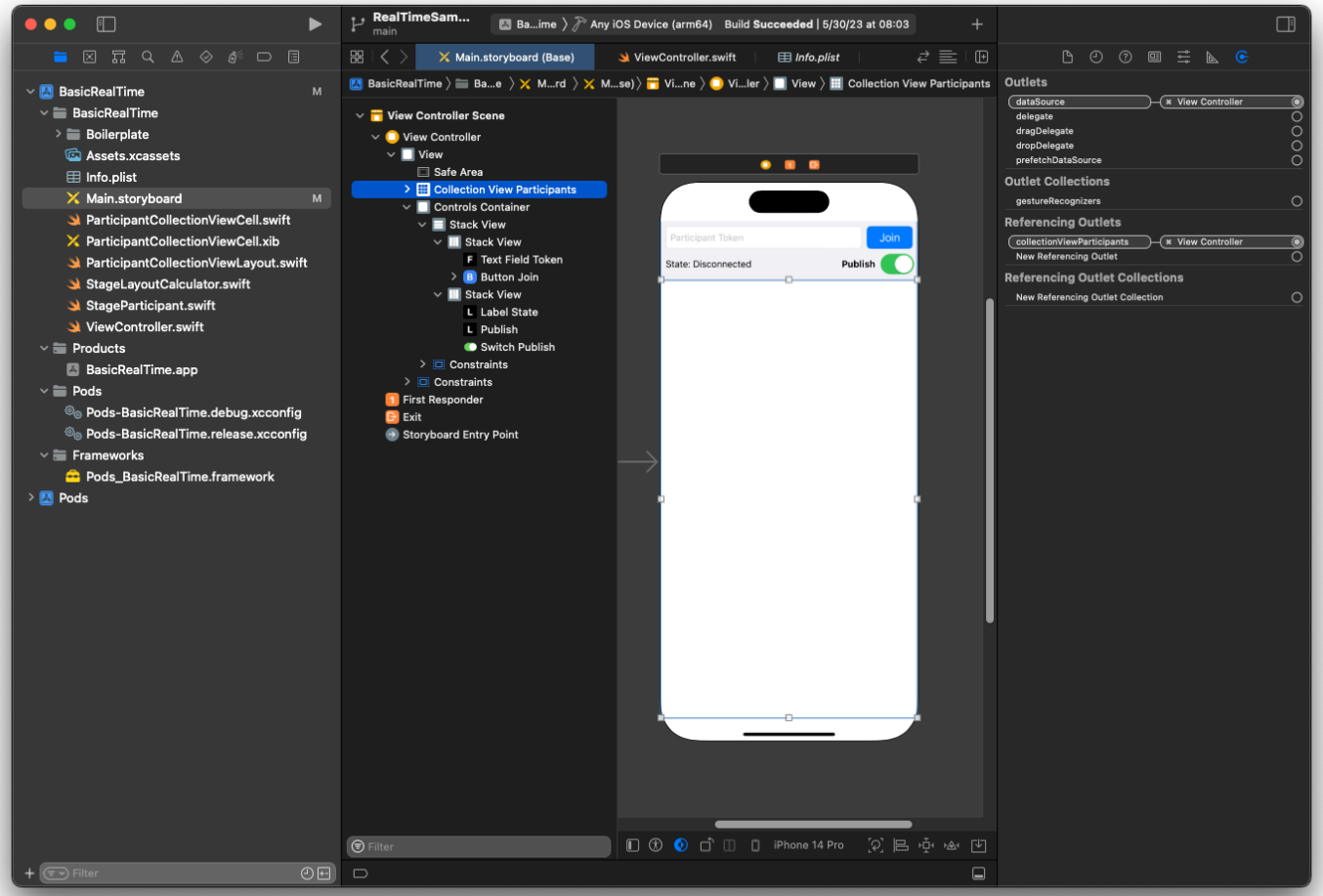
UIStackViews は残りのビューのレイアウトを処理します。3 つの UI StackViewsすべてについて、アラインメント とディストリビューション として Fill を使用します。



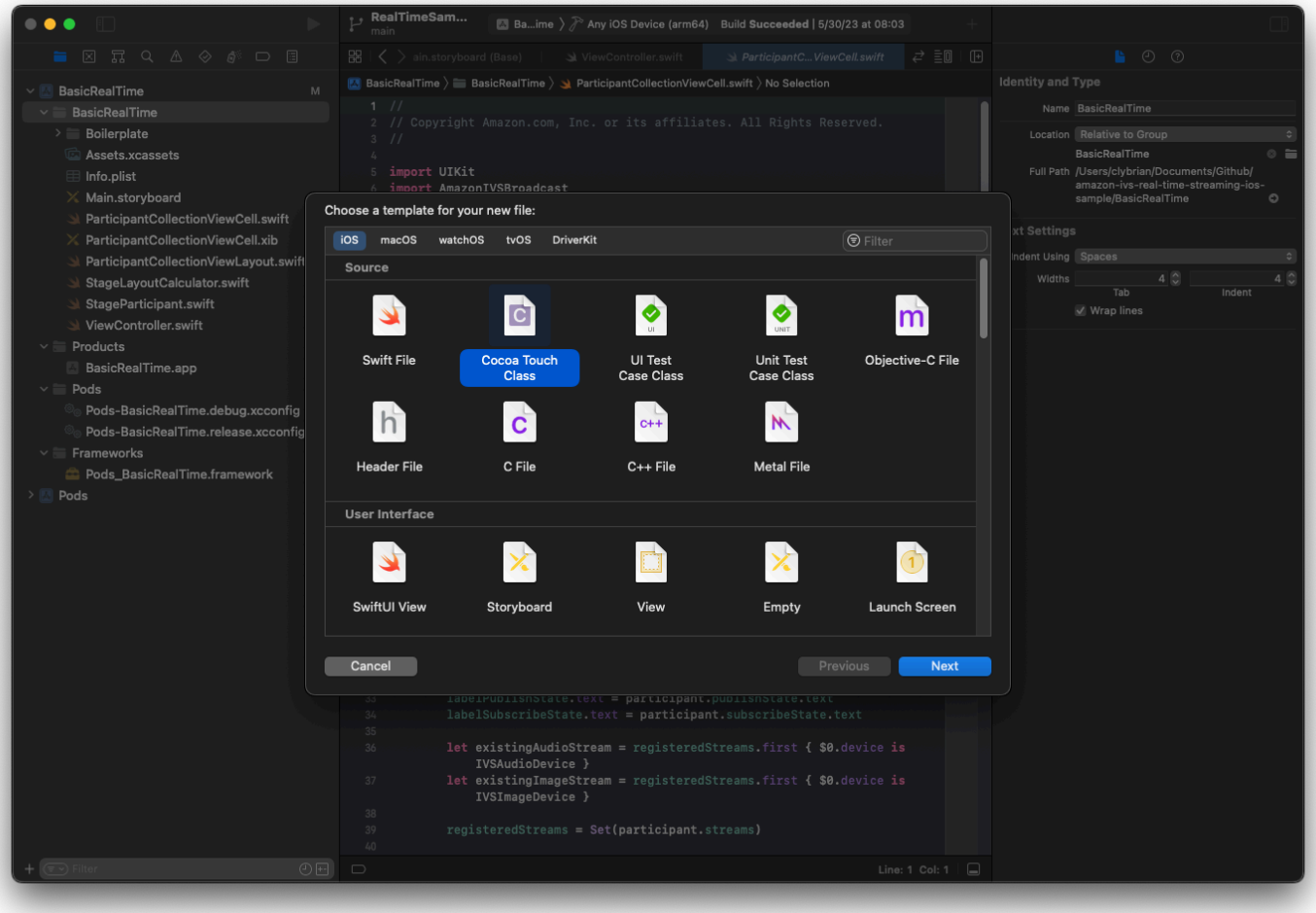
最後に、これらのビューを ViewController にリンクしましょう。上から、次のビューをマッピングします。

- [テキストフィールド結合] を textFieldToken にバインドします。
- [ボタン結合] を buttonJoin にバインドします。
- [ラベル状態] を labelState にバインドします。
- [公開の切り替え] を switchPublish にバインドします。
- [コレクションビュー参加者] を collectionViewParticipants にバインドします。

また、この時間を利用して、[コレクションビュー参加者] 項目の dataSource を所有する ViewController に設定します。



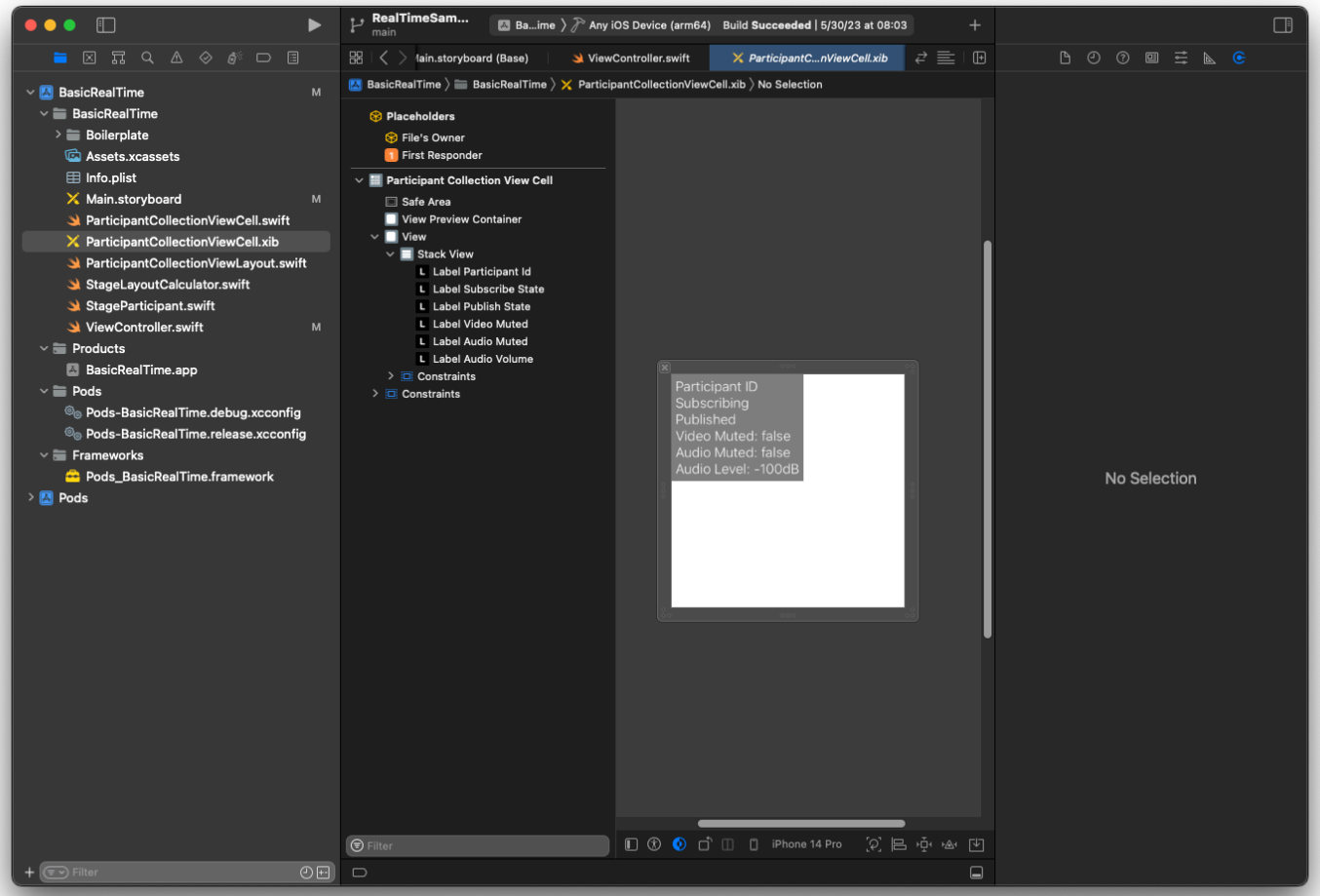
次に、参加者をレンダリングする `UICollectionViewCell` サブクラスを作成します。まず、新しい[Cocoa Touch Class] ファイルを作成します。



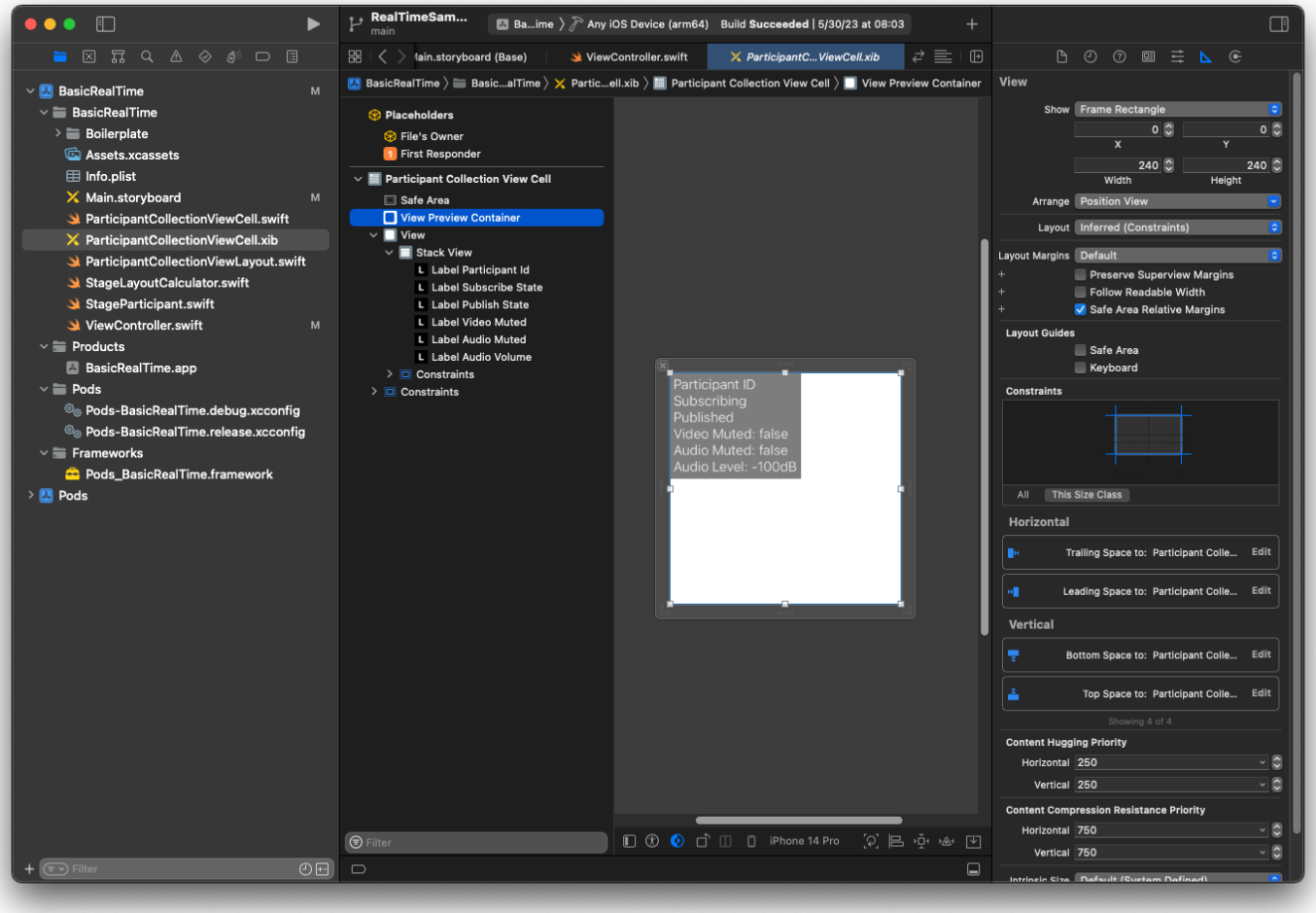
ParticipantUICollectionViewCell と名前を付けて、Swift 内にある UICollectionViewCell のサブクラスにします。もう一度、Swift ファイルから始めます。リンクする @IBOutlets を作成します。

```
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!
```

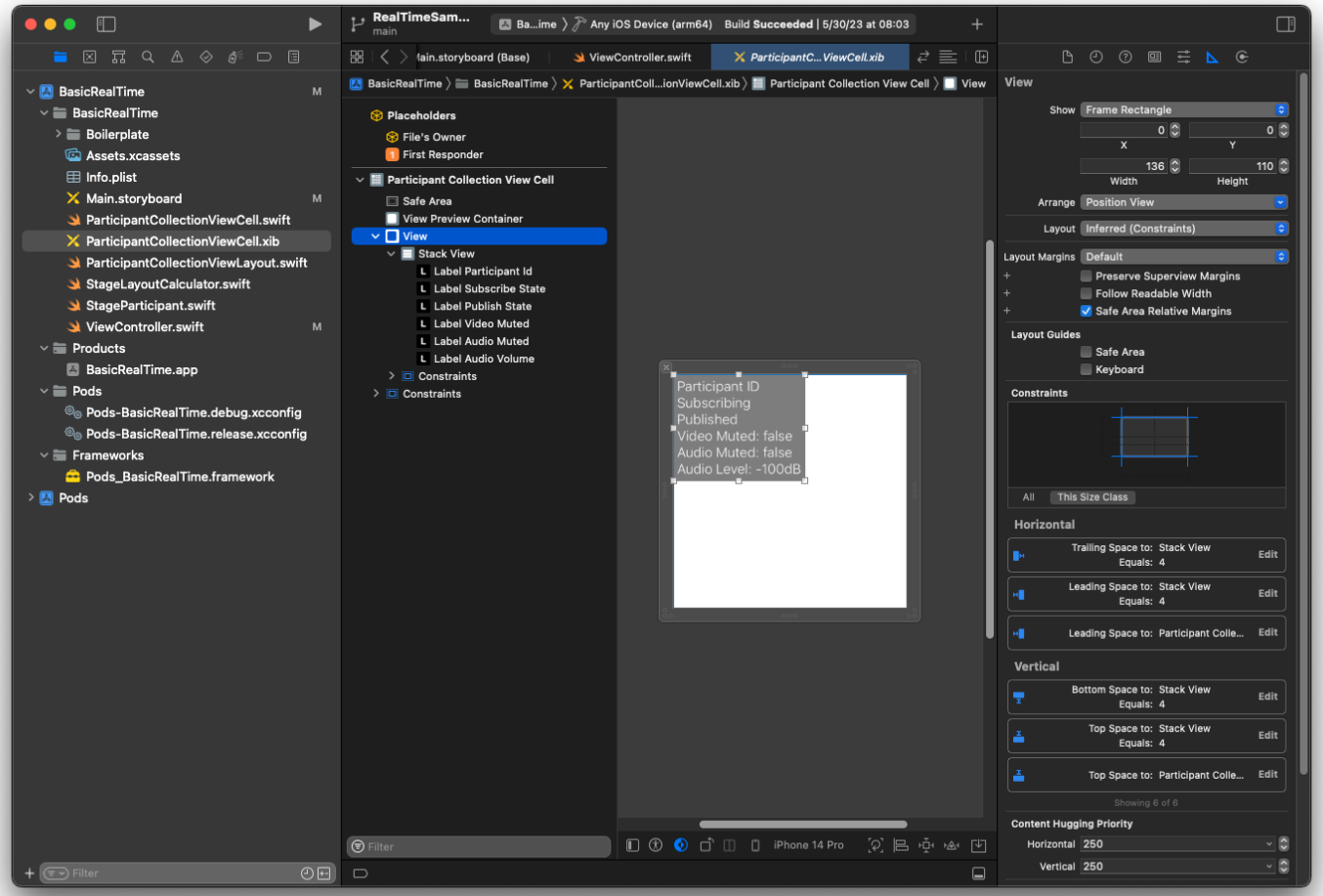
関連付けられている XIB ファイルに、次のビュー階層を作成します。



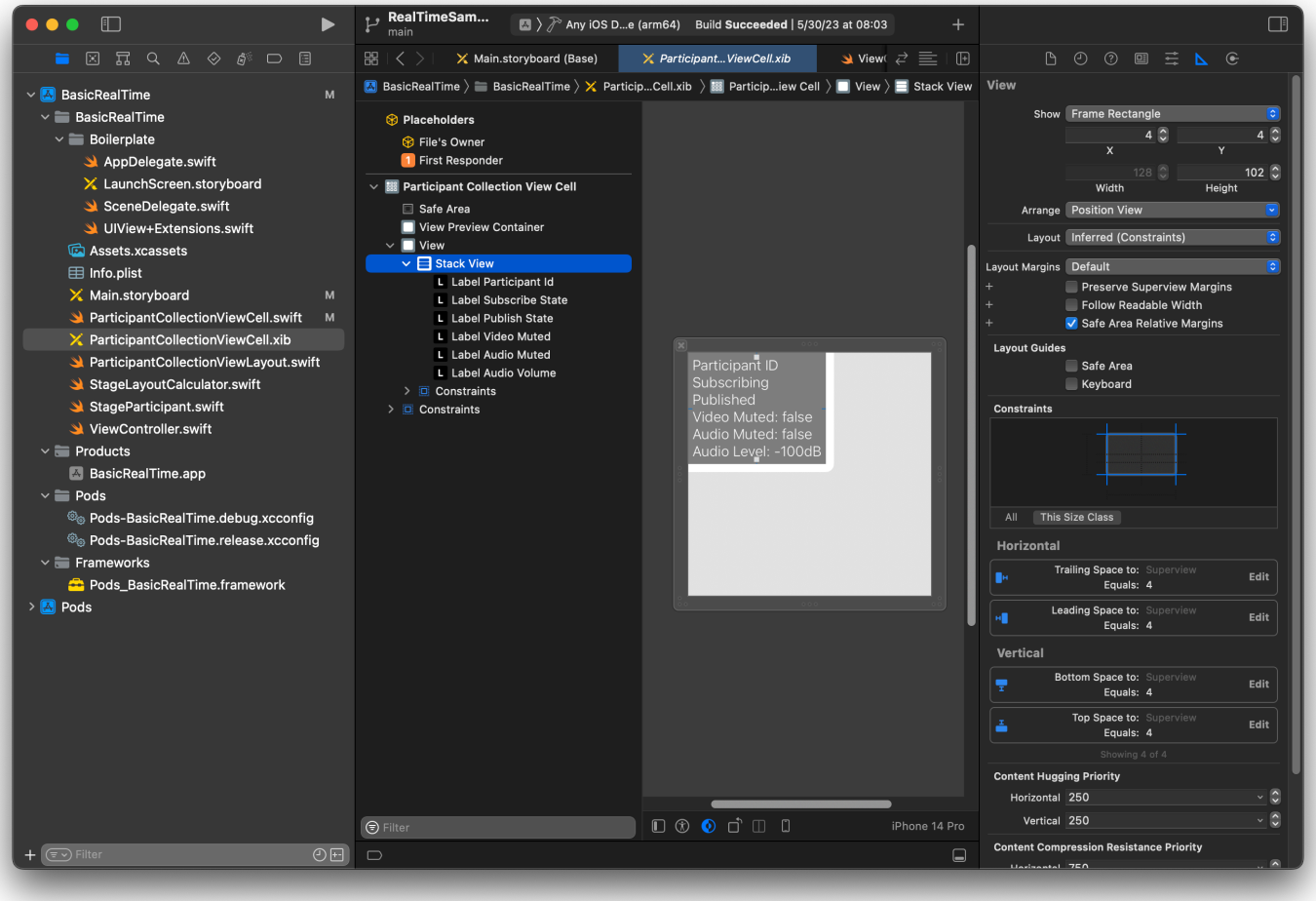
では AutoLayout、3 つのビューを再度変更します。最初のビューは [ビュープレビューコンテナ] です。[末尾]、[先頭]、[上]、[下] を [参加者コレクションビューセル] に設定します。



2 番目のビューは [ビュー] です。[先頭] および [上] を [参加者コレクションビューセル] に設定して、値を 4 に変更します。



3 番目のビューは [スタックビュー] です。[末尾]、[先頭]、[上]、[下] を [Superview] に設定して、値を 4 に変更します。



アクセス許可とアイドルタイマー

ViewController に戻り、アプリケーションの使用中にデバイスがスリープ状態にならないように、システムのアイドルタイマーを無効にします。

```
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
}
```

次に、システムにカメラとマイクへのアクセス許可をリクエストします。

```
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)
    }
}
```

アプリの状態

collectionViewParticipants を、先ほど作成したレイアウトファイルで設定する必要があります。

```
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")
}
```

各参加者を表すために、StageParticipant という名の単純な構造体を作成します。これは ViewController.swift ファイルに記述することができますが、新しいファイルを作成してもかまいません。

```
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
    }
}
```

これらの参加者を追跡するために、参加者の配列を ViewController にプライベートプロパティとして保持します。

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

このプロパティは、先ほどストーリーボードからリンクされた UICollectionViewDataSource に使用します。

```
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'")
        }
    }

}
```

ステージに参加する前に自分のプレビューを確認できるように、ローカル参加者をすぐに作成します。

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

これにより、アプリが実行されるとすぐに、ローカル参加者を表す参加者セルがレンダリングされます。

ユーザーには、ステージに参加する前に自分自身を確認する機能が必要です。このため、次に、先程のアクセス許可処理コードから呼び出されれる setupLocalUser() メソッドを実装します。カメラとマイクのリファレンスは IVSLocalStageStream オブジェクトとして保存します。。

```
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)
}
```

ここでは、SDK を通してデバイスのカメラとマイクを検出し、それらをローカルの streams オブジェクトに格納し、その後、最初の参加者 (先ほど作成したローカル参加者) の streams 配列を streams に割り当てています。最後に、0 の index とupdated の changeType で participantsChanged を呼び出します。この関数は、適切なアニメーション付きで UICollectionView を更新するヘルパー関数です。以下のようになります。

```
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)])
    }
}
```

cell.set については後で説明しますが、ここで参加者に基づいてセルの内容をレンダリングします。

ChangeType は単純な列挙型です。

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

```
}
```

最後に、ステージが接続されているかどうかを追跡しましょう。追跡にはシンプルな bool を使用します。これ自身が更新されると、UI も自動的に更新されます。

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

ステージ SDK の実装

リアルタイム機能には、ステージ、ストラテジー、レンダラーという 3 つのコアコンセプトがあります。設計目標は、実際に動作する製品を構築するのに必要となるクライアント側ロジックの量を最小限に抑えることです。

IVSStageStrategy

IVSStageStrategy の実装は簡単です。

```
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
    }
}
```

簡単に説明すると、公開スイッチが「オン」の位置にある場合にのみ公開し、公開する場合には、以前に収集したストリームが公開されます。このサンプルでは、常に他の参加者をサブスクライブして、オーディオとビデオの両方を受信しています。

IVSStageRenderer

IVSStageRenderer の実装も比較的簡単ですが、関数の数が多いことから含まれるコードの数がかなり多くなっています。このレンダラーの全体的なアプローチは、SDK から参加者の変更を通知されたときに participants 配列を更新するというものです。ローカル参加者が、参加する前にカメラのプレビューを確認できるように自分たちで管理することにしたため、一部のシナリオではローカル参加者の扱い方が異なる場合があります。

```
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)
    }
}
```

このコードでは、拡張機能を使って接続状態をわかりやすいテキストへと変換しています。

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

カスタム UI の実装CollectionViewLayout

異なる人数の参加者をレイアウトするのは複雑です。親ビューのフレーム全体を占めるようにしたいものの、各参加者の設定を個別に処理するのは面倒です。これを簡単にするために、UICollectionViewLayout の実装について順を追って説明します。

別の新しいファイル ParticipantCollectionViewLayout.swift を作成します。これを使用して UICollectionViewLayout を拡張します。このクラスは、StageLayoutCalculator という別のクラスを使用します。これについては後ほど説明します。このクラスは、各参加者の計算されたフレーム値を受け取り、その後、必要な UICollectionViewLayoutAttributes オブジェクトを生成します。

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

もっと重要となるのは StageLayoutCalculator.swift クラスです。このクラスは、フローベースの行/列レイアウトの参加者数に基づいて、各参加者のフレームを計算するように設計されています。各行の高さは他の行と同じですが、列の幅は行ごとに異なる場合があります。この動作をカスタマイズする方法については、layouts 変数の上のあるコードコメントを参照してください。

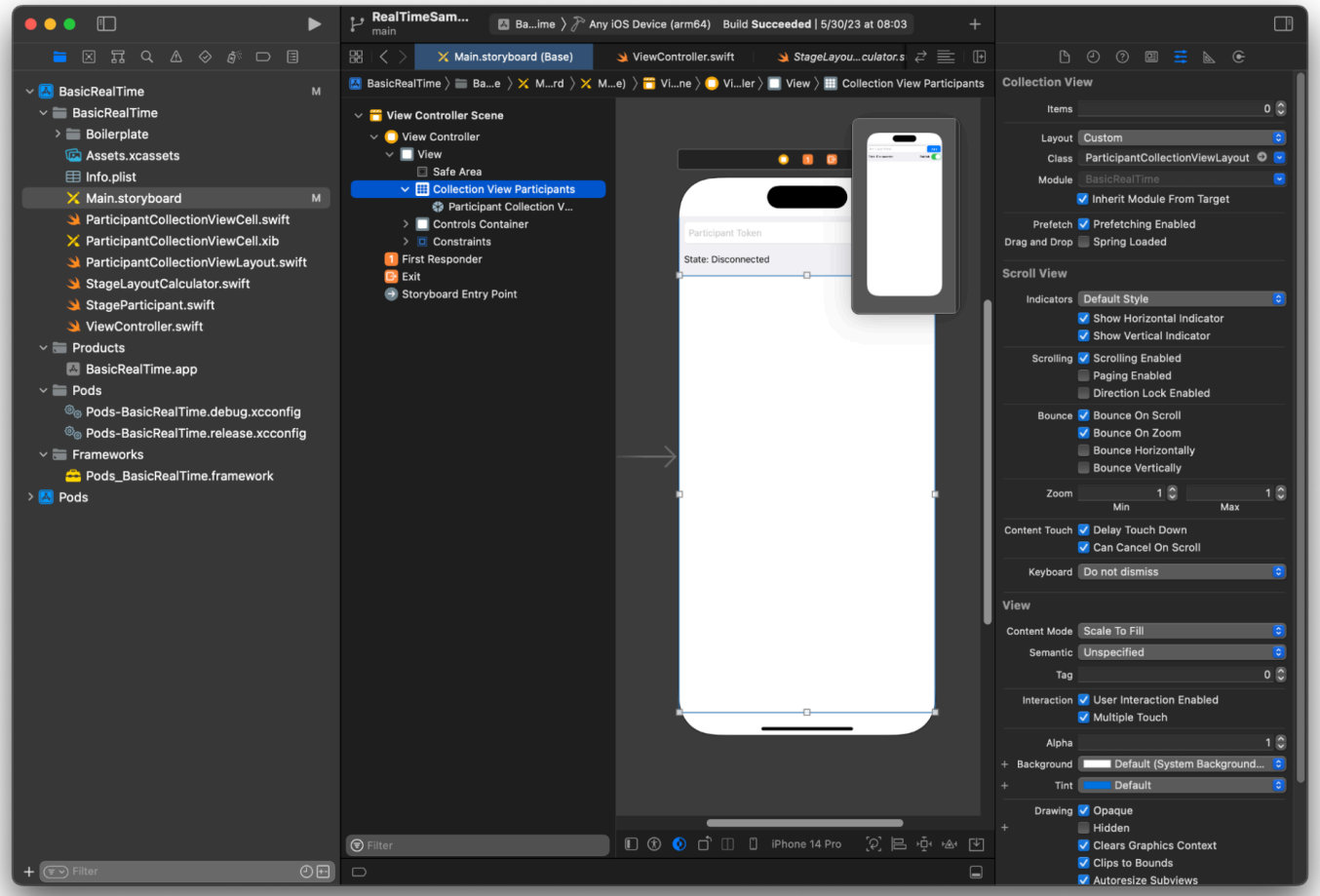
```
import Foundation
import UIKit
```

```
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
    }

}
```

Main.storyboard に戻り、UICollectionView のレイアウトクラスを先ほど作成したクラスに設定してください。



UI アクションの接続

もうすぐ完了です。作成する必要のある IBActions がいくつかあります。

まず、参加ボタンを処理しましょう。レスポンスは connectingOrConnected の値によって異なります。すでに接続されている場合は、ステージを離れるだけです。接続されていない場合は、トークン UITextField からテキストを読み取り、そのテキストを使用して新しい IVSStage を作成します。次に、ViewController を strategy、errorDelegate、IVSStage のレンダラーとして追加し、最後にステージを非同期で結合します。

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

もう 1 つの接続する必要がある UI アクションは、公開スイッチです。

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

参加者のレンダリング

最後に、SDK から受け取ったデータを、先ほど作成した参加者セルにレンダリングする必要があります。UICollectionView ロジックはすでに完了しているので、ParticipantCollectionViewCell.swift の set API を実装するだけです。

まず empty 関数を追加した後に、1 つずつ見ていきましょう。

```
func set(participant: StageParticipant) {

}
```

まず、簡易状態、参加者 ID、公開状態、サブスクライブ状態を処理します。これらについては、UILabels を直接更新します。

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

公開列挙型とサブスクライブ列挙型のテキストプロパティは、ローカル拡張機能から取得します。

```
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()
        }
    }
}
```

次に、オーディオとビデオのミュート状態を更新します。ミュート状態を取得するには、streams 配列から IVSImageDevice と IVSAudioDevice を見つける必要があります。パフォーマンスを最適化するために、最後にアタッチされたデバイスを記憶しています。

```
// 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)"
```

最後に、imageDevice のプレビューをレンダリングして、audioDevice からのオーディオ状態を表示しましょう。

```
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"
}
```

作成する必要のある最後の関数は updatePreview() です。この関数で参加者のプレビューをビューに追加します。

```
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)
        }
    }
```

```
}
```

上記では、UIView のヘルパー関数を使用して、サブビューの埋め込みを容易にしています。

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

Amazon IVS Real-Time Streaming のモニタリング

ステージセッションとは

ステージセッションは、最初の参加者がステージに参加したときに始まり、最後の参加者がステージへの公開を停止した数分後に終了します。ステージセッションは、イベントと参加者を有効期間の短いセッションに分けることで、有効期間の長いステージのデバッグに役立ちます。

ステージセッションと参加者の表示

コンソールでの手順

1. Amazon IVS コンソールを開きます。

   (AWS マネジメントコンソールから Amazon IVS コンソールにアクセスすることもできます。)
2. ナビゲーションペインで [ステージ] を選択します。(ナビゲーションペインが折りたたまれている場合は、まずハンバーガーアイコンを選択して開きます。)
3. ステージを選択して、その詳細ページに移動します。
4. [ステージセッション] セクションが表示されるまでページを下にスクロールし、ステージセッションを選択して詳細ページを表示します。
5. セッションの参加者を表示するには、[参加者] セクションが表示されるまで下にスクロールし、参加者を選択して、Amazon CloudWatch メトリクスのグラフを含む詳細ページを表示します。

参加者にイベントを表示

イベントは、ステージに参加したり、ステージに公開しようとしたときにエラーが発生したりするなど、ステージ内の参加者のステータスが変化したときに送信されます。すべてのエラーがイベントを引き起こすわけではありません。たとえば、クライアント側のネットワークエラーやトークン署名エラーはイベントとして送信されません。クライアントアプリケーションでこれらのエラーを処理するには、IVS Broadcast SDK を使用します。

コンソールでの手順

1. 上記の手順に従って、参加者の詳細ページに移動します。

2. [イベント] セクションが表示されるまで下にスクロールします。参加者イベントの順序付けされたリストが表示されます。参加者に送信されるイベントの詳細については、「Amazon Interactive Video Service で Amazon EventBridge を使用する」を参照してください。

CLI の手順

AWS CLI によるステージセッションイベントへのアクセスは高度なオプションであり、まず CLI をダウンロードしてマシン上で設定する必要があります。詳細については、AWS Command Line Interface のユーザーガイドを参照してください。

1. ステージセッションを一覧表示してステージセッションを検索します。

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

2. ステージセッションの参加者を一覧表示して参加者を検索します。

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

3. ステージセッションと参加者のイベントを一覧表示します。

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

`list-participant-events` 呼び出しのサンプルレスポンスを次に示します。

```
{
    "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"
        }
    ]
}
```

CloudWatch メトリクスへのアクセス

CloudWatch メトリクスを使用するには、Web 1.5.0 以降、Android 1.12.0 以降、または iOS 1.12.0 以降の IVS Broadcast SDK バージョンが必要です。

CloudWatch コンソールでの手順

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. サイドナビゲーションで、[メトリクス] ドロップダウンをクリックし、[すべてのメトリクス] を選択します。
3. [参照] タブで、左側のラベルなしのドロップダウンを使用して、チャネルが作成された「ホーム」リージョンを選択します。リージョンの詳細については、「グローバルソリューション、リージョナルコントロール」を参照してください。対応するリージョンの一覧については、「AWS 全般のリファレンス」のAmazon IVS のページを参照してください。
4. [参照] タブの下部で [IVSRealtime] 名前空間を選択します。
5. 次のいずれかを行います:
   a. 検索バーに、リソース ID (ARN の一部、`arn:::ivs:stage/<resource id>`) を入力します。

      次に [IVSRealTime]、[ステージメトリクス] の順に選択します。

   b. [AWS の名前空間]に [IVSRealTime] が選択可能なサービスとして表示されたら選択します。これは、Amazon IVS Real-Time Streaming を使用して、Amazon CloudWatch にメトリクスを送信している場合に表示されます。([IVSRealTime]がリストに表示されない場合、Amazon IVS メトリクスはありません。)

次に、必要に応じてディメンショングループを選択します。使用可能なディメンションは、以下の「CloudWatch メトリクス」にリストされています。

6. グラフに追加するメトリクスを選択します。利用可能なメトリックスは、以下の「CloudWatch メトリクス」にリストされています。

ストリームセッションの詳細ページで [CloudWatch で表示] ボタンを選択して、ストリームセッションの CloudWatch グラフにアクセスすることもできます。

CLI の手順

AWS CLI を使用してメトリクスにアクセスすることもできます。そのためには、まず CLI をマシンにダウンロードして設定する必要があります。詳細については、「AWS Command Line Interface のユーザーガイド」を参照してください。

次に、AWS CLI を使用して Amazon IVS Real-Time Streaming メトリクスにアクセスするために、次の操作を実行します。

- コマンドプロンプトで、次のコマンドを実行します。

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

詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch メトリクスの使用」を参照してください。

CloudWatch メトリクス: IVS リアルタイムストリーミング

Amazon IVS は、AWS/IVSRealTime 名前空間で以下のメトリクスを提供します。

CloudWatch メトリクスを使用するには、Web Broadcast SDK 1.5.2 以降を使用する必要があります。

ディメンションには以下の有効な値があります。

- `Stage` ディメンションはリソース ID (ARN の一部、`arn:::stage/<resource id>`) です。
- `Participant` ディメンションは `participantID` です。
- `SimulcastLayer` は、video の `MediaType` では hi、mif、low、または no-rid で、audio の `MediaType` では disabled です。この値は空欄でも構いません。

- MediaType ディメンションは「ビデオ」または「オーディオ」(文字列) です。

メトリクス	ディメンション	説明
DownloadPacketLoss	Stage	各サンプルは、特定のサブスクライバーが IVS サーバーからのダウンロード中に失ったパッケージの割合を表します。 単位: パーセント 有効な統計: 平均、最大、最小 — 設定された間隔での失われたパケットの平均数、最大数、または最小数。
DownloadPacketLoss	Stage,Participant	パブリッシャーでもあるサブスクライバーを対象に、参加者別に DownloadPacketLoss をフィルタリングします。サンプルは、サブスクライバーが IVS サーバからのダウンロード中に失ったパッケージの割合を表します。サンプルは、参加者がパブリッシャーでもある場合にのみ送信されます。 単位: パーセント 有効な統計: 平均、最大、最小 — 設定された間隔でのドロップされたフレームの平均数、最大数、または最小数。
DroppedFrames	Stage	各サンプルは、特定のサブスクライバーによってドロップされたフレームの割合を表します。 単位: パーセント 有効な統計: 平均、最大、最小 — 設定された間隔でのドロップされたフレームの平均数、最大数、または最小数。
DroppedFrames	Stage,Participant	パブリッシャーでもあるサブスクライバーを対象に、参加者別に DroppedFrames をフィルタリングします。サンプルは、サブスクライブしている参加者とステージ内のすべてのパブリッシャーの間でドロップされた

メトリクス	ディメンション	説明
		フレームの割合を表します。サンプルは、参加者がパブリッシャーでもある場合にのみ送信されます。 単位: パーセント 有効な統計: 平均、最大、最小 — 設定された間隔でのドロップされたフレームの平均数、最大数、または最小数。
PublishBitrate	Stage	発行されるサンプルは、特定のパブリッシャーがビデオデータとオーディオデータの両方を送信する合計レート（すべてのサイマルキャストレイヤーの合計）を表します。 単位: ビット/秒 有効な統計: 平均、最大、最小 — 設定された間隔でのビットレートの平均数、最大数、または最小数。
PublishBitrate	Stage, Participant, SimulcastLayer, MediaType	参加者、サイマルキャストレイヤー、メディアタイプで PublishBitrate をフィルタリングします。サイマルキャストレイヤー ID は Broadcast SDK によって設定されます。サイマルキャストを無効にすると、このレイヤー ID は「無効」に設定されます。メディアタイプはビデオまたはオーディオです。 単位: ビット/秒 有効な統計: 平均、最大、最小 — 設定された間隔でのビットレートの平均数、最大数、または最小数。
Publishers	Stage	ステージに公開する参加者の数。 単位: 個 有効な統計: 平均、最大、最小

メトリクス	ディメンション	説明
PublishResolution	Stage, Participant, SimulcastLayer, MediaType	フレームの幅または高さのうち小さい方のピクセル数。例えば、サイズが 1920 x 1080 のランドスケープフレームの場合、PublishResolution は 1080 です。サイズが 720 x 1280 のポートレートフレームの場合、PublishResolution は 720 です。 単位: 個 有効な統計: 平均、最大、最小
SubscribeBitrate	Stage	発行されるサンプルは、特定のサブスクライバーがビデオデータとオーディオデータの両方を受信する合計レートを表します。 単位: ビット/秒 有効な統計: 平均、最大、最小 — 設定された間隔でのビットレートの平均数、最大数、または最小数。
SubscribeBitrate	Stage,Participant, MediaType	パブリッシャーでもあるサブスクライバーを対象に、参加者別に SubscribeBitrate をフィルタリングします。サンプルは、特定のサブスクライバーが特定の MediaType を受信しているビットレートを表します。サンプルは、サブスクライブしている参加者が公開している間のみ出力されます。 単位: ビット/秒 有効な統計: 平均、最大、最小 — 設定された間隔でのビットレートの平均数、最大数、または最小数。

メトリクス	ディメンション	説明
Subscribers	Stage	ステージにサブスクライブされる参加者の数。公開とサブスクライブをアクティブに行っている参加者は、パブリッシャーとサブスクライバーの両方としてカウントされます。 単位: 個 有効な統計: 平均、最大、最小

IVS Broadcast SDK (リアルタイムストリーミング)

Amazon Interractive Video Searvice (IVS) リアルタイムストリーミングブロードキャスト SDK は、Amazon IVS を使用してアプリケーションを構築するデベロッパー向けのものです。この SDK は、Amazon IVS のアーキテクチャを活用するように設計されており、Amazon IVS と共に、継続的に改善され、新機能が提供されます。ネイティブのブロードキャスト SDK として、アプリケーションおよびユーザーがアプリケーションにアクセスするデバイスに対してパフォーマンスへの影響を最小限に抑えるように設計されています。

Broadcast SDK はビデオの送信と受信の両方に使用されることに注意してください。ホストとビューアーには同じ SDK を使用します。個別のプレーヤー SDK は必要ありません。

アプリケーションでは、Amazon IVS Broadcast SDK の主な機能を活用できます。

- 高品質ストリーミング — ブロードキャスト SDK は、高品質のストリーミングをサポートします。カメラからビデオをキャプチャし、最大 720p でエンコードします。
- 自動ビットレート調整 — スマートフォンユーザーは移動するため、ブロードキャストの過程でネットワークの状況が変わることがあります。Amazon IVS Broadcast SDK は、変化するネットワーク状況に対応するために、動画のビットレートを自動的に調整します。
- 縦向きと横向きのサポート — ユーザーがデバイスをどのように持っているかに関係なく、画像は正しい向きで適切に拡大縮小されて表示されます。Broadcast SDK は、ポートレートとランドスケープの両方のキャンバスサイズをサポートします。ユーザーが設定した向きからデバイスを回転させると、アスペクト比が自動的に調整されます。
- セキュアなストリーミング — ユーザーのブロードキャストは、 TLS を使用して暗号化されるため、ストリームを安全に保つことができます。
- 外部オーディオデバイス — Amazon IVS Broadcast SDK は、オーディオジャック、USB、および Bluetooth SCO 外部マイクをサポートしています。

プラットフォームの要件

ネイティブプラットフォーム

プラットフォーム	サポートされているバージョン
Android	9.0 以降 -- バージョン 5.0 でビルドすることはできますが、リアルタイムストリーミング機能は使用できないことに注意してください。
iOS	14 以降

IVS は、少なくとも 4 つの iOS メジャーバージョンと 6 つの Android メジャーバージョンをサポートしています。現在のサポート対象バージョンは、これらの最小値よりも多い可能性があります。メジャーバージョンがサポートされなくなる場合は、少なくとも 3 か月前に SDK リリースノートでお客様にお知らせします。

デスクトップブラウザ

ブラウザ	サポートされているプラットフォーム	サポートされているバージョン
Chrome	Windows、macOS	2 つのメジャーバージョン (最新バージョンと 1 つ前のバージョン)
Firefox	Windows、macOS	2 つのメジャーバージョン (最新バージョンと 1 つ前のバージョン)
Edge	(Windows 8.1 以降)	2 つのメジャーバージョン (最新バージョンと 1 つ前のバージョン) Edge Legacy は除く
Safari	macOS	2 つのメジャーバージョン (最新バージョンと 1 つ前のバージョン)

モバイルブラウザ (iOS および Android)

ブラウザ	サポートされているプラットフォーム	サポートされているバージョン
Chrome	iOS、Android	2 つのメジャーバージョン (最新バージョンと 1 つ前のバージョン)
Firefox	Android	2 つのメジャーバージョン (最新バージョンと 1 つ前のバージョン)
Safari	iOS	2 つのメジャーバージョン (最新バージョンと 1 つ前のバージョン)

既知の制限事項

- どのモバイルデバイスでも、動画の乱れや画面の暗転の問題があるため、4 人以上の参加者が同時に公開/サブスクライブすることはお勧めしません。参加者をそれより多くする必要がある場合は、オーディオのみの公開とサブスクライブを設定します。
- パフォーマンス上の問題やクラッシュの可能性を考慮して、ステージを合成して Android Mobile Web のチャネルに配信することはお勧めしません。ブロードキャスト機能が必要な場合は、IVS リアルタイムストリーミングの Android Broadcast SDK を統合してください。

ウェブビュー

Web Broadcast SDK は、ウェブビューやウェブライク環境 (テレビ、家庭用ゲーム機など) をサポートしていません。モバイル実装については、Android 向けおよび iOS 向けの「IVS Broadcast SDK ガイド (リアルタイムストリーミング)」を参照してください。

必要なデバイスのアクセス

Broadcast SDK では、デバイス内蔵のカメラとマイクと、Bluetooth、USB、またはオーディオジャックを介して接続されているカメラとマイクにアクセスする必要があります。

サポート

ブロードキャスト SDK は継続的に改良しています。利用可能なバージョンと修正済みの問題については Amazon IVS リリースノートを参照してください。必要な場合、サポートに連絡する前にお使いのプレイヤーのバージョンを更新し、問題が解決するかどうか確認してください。

バージョニング

Amazon IVS Broadcast SDK は、セマンティックバージョニングを使用しています。

以下の解説は、次を前提としています。

- 最新リリースは 4.1.3。
- 1 つ前のメジャーバージョンの最新リリースは 3.2.4。
- バージョン 1.x の最新リリースは 1.5.6.

最新バージョンのマイナーリリースとして、下位互換性のある新機能が追加されています。この場合、次回の新機能のセットは、バージョン 4.2.0 として追加されます。

下位互換性のあるマイナーなバグ修正が、最新バージョンのパッチリリースとして追加されています。ここでは、次回のマイナーなバグ修正のセットは、バージョン 4.1.4 として追加されます。

下位互換性のあるメジャーなバグ修正は異なる方法で処理されます。これらはいくつかのバージョンに追加されています。

- 最新バージョンのパッチリリース。こちらは、バージョン 4.1.4 です。
- 1 つ前のマイナーバージョンのパッチリリース。こちらは、バージョン 3.2.5 です。
- 最新バージョン 1.x リリースのパッチリリース。こちらは、バージョン 1.5.7 です。

メジャーなバグ修正は、Amazon IVS 製品チームによって定義されています。典型的な例に、重要なセキュリティ更新のほか、お客様に必要な選別された修正があります。

注: 上記の例では、リリースされたバージョンの数字は、連番でインクリメントされています(4.1.3 → 4.1.4、など)。実際は、1 つ以上のパッチ番号が内部に残り、リリースされないままになることもあります。そのため、リリースされたバージョンは 4.1.3 から (例えば) 4.1.6 に増えることもあります。

IVS Broadcast SDK: Web ガイド (リアルタイムストリーミング)

IVS リアルタイムストリーミング Web Broadcast SDK は、デベロッパー向けに、Web 上でインタラクティブかつリアルタイムの体験を構築するためのツールを提供します。この SDK は、Amazon IVS を使用してウェブアプリケーションを構築するデベロッパー向けです。

Web Broadcast SDK を使用して、参加者はビデオを送受信できます。SDK は、次の操作をサポートします。

- ステージに参加する
- ステージ内の他の参加者にメディアを公開する
- ステージ内の他の参加者のメディアをサブスクライブする
- ステージに公開されたビデオとオーディオを管理および監視する
- 各ピア接続の WebRTC 統計を取得
- IVS 低遅延ストリーミング Web Broadcast SDK からのすべての操作

Web Broadcast SDK の最新バージョン： 1.8.0 (リリースノート)

リファレンスドキュメント： Amazon IVS Web Broadcast SDK で使用できる最も重要な方法については、「https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference」を参照してください。SDK の最新バージョンが選択されていることを確認してください。

サンプルコード: SDK をすぐに使い始めるには、以下のサンプルの利用が適しています。

- HTML と JavaScript
- React

プラットフォーム要件: サポートされているプラットフォームのリストについては、「Amazon IVS Broadcast SDK」を参照してください。

開始方法

インポート

リアルタイムのビルディングブロックは、ルートブロードキャストモジュールとは別の名前空間に配置されています。

スクリプトタグを使用する

同じスクリプトインポートを使用すると、以下の例で定義されているクラスと列挙型がグローバル IVSBroadcastClient オブジェクトにあります。

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

npmを使う

クラス、列挙型、型はパッケージモジュールからインポートすることもできます。

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

必要なアクセス許可

アプリケーションは、ユーザーのカメラとマイクへのアクセス許可をリクエストする必要があります。また、HTTPS で提供される必要があります。(これは Amazon IVS に特有ではなく、カメラやマイクにアクセスが必要なすべてのウェブサイトに必要です。)

オーディオおよびビデオデバイス両方のアクセス許可をリクエストし、キャプチャする方法を示す関数の例を次に示します。

```
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.');
    }
}
```

詳細については、「Permissions API」およびMediaDevicesgetUserMedia「()」を参照してください。

利用可能なデバイスのリストを表示する

キャプチャできるデバイスを確認するには、ブラウザの.MediaDevicesenumerateDevices () メソッドをクエリします。

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

MediaStream デバイスから を取得する

使用可能なデバイスのリストを獲得すると、任意の数のデバイスからストリームを取得できます。例えば、カメラからストリームを取得する getUserMedia() メソッドを利用できます。

ストリームをキャプチャするデバイスを指定する場合は、メディア制約の audio または video セクションで deviceId を明示的に設定できます。または、deviceId を省略して、ブラウザのプロンプトからユーザーにデバイスを選択させることもできます。

width および height の制約を使用して、理想的なカメラの解像度を指定することもできます。(これらの制約について詳しくは、こちらをご覧ください。) SDK では、ブロードキャストの最大解像度に対応する幅および高さの制約が自動的に適用されます。しかし、ソースを SDK に追加した後ソースのアスペクト比が変更されないよう、これらもお客様ご自身で適用することをお勧めします。

リアルタイムストリーミングの場合は、メディアの解像度が 720p に制限されていることを確認してください。具体的には、幅getUserMediaと高さの と getDisplayMedia の制約値を乗算すると、921600 (1280*720) を超えることはできません。

```
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 },
});
```

公開とサブスクライブ

概念

リアルタイム機能の根底には、ステージ、ストラテジー、イベントという 3 つのコアコンセプトがあります。設計目標は、実際に動作する製品を構築するのに必要となるクライアント側ロジックの量を最小限に抑えることです。

ステージ

Stage クラスは、ホストアプリケーションと SDK 間の主要な相互作用のポイントです。これはステージそのものを表し、ステージへの参加とステージからの退出に使用されます。ステージの作成と参加には、コントロールプレーンからの有効で有効期限内のトークン文字列 (token として表示) が必要です。ステージへの参加と退出は簡単です。

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

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

stage.leave();
```

方針

StageStrategy インターフェースは、ホストアプリケーションがステージの望ましい状態を SDK に伝える方法を提供しま
す。shouldSubscribeToParticipant、shouldPublishParticipant、stageStreamsToPublish の 3 つの関数を実装する必要があります。以下で、すべて説明します。

定義済みのストラテジーを使用するには、それを Stage コンストラクターに渡します。以下は、参加者の Web カメラをステージに公開し、すべての参加者にサブスクライブするというストラテジーを使用したアプリケーションの完全な例です。必要な各ストラテジー機能の目的は、以下のセクションで説明します。

```
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();
```

参加者へのサブスクライブ

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

リモート参加者がステージに参加すると、SDK はその参加者に対して希望するサブスクリプションの状態についてホストアプリケーションに問い合わせます。使用できるオプションは NONE、AUDIO_ONLY、および AUDIO_VIDEO です。この関数の値を返す場合、ホストアプリケーションは公開の状態、現在のサブスクリプションの状態、またはステージ接続の状態を考慮する必要はありません。AUDIO_VIDEO が返された場合、SDK はリモート参加者が公開するまで待ってからサブスクライブし、プロセス全体でイベントを作成してホストアプリケーションを更新します。

次に示すのは実装の例です。

```
const strategy = {

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

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

これは、ビデオチャットアプリケーションなど、すべての参加者が互いに常に可視化されているホストアプリケーション向けのの完全な実装です。

より高度な実装も可能です。ParticipantInfo の userInfo プロパティを使用して、サーバーが提供する属性に基づいて、参加者に対して選択的にサブスクライブできます。

```
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
}
```

これを使用すると、モデレーターは、自身は視聴の対象とならずに、すべてのゲストを監視できるステージを作ることができます。ホストアプリケーションでは、追加のビジネスロジックを使用して、モデレータがお互いを見えるようにしても、ゲストには見えないようにすることができます。

公開

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

ステージに接続すると、SDK はホストアプリケーションにクエリを実行し、特定の参加者を公開とすべきかどうかを確認します。これは、提供されたトークンに基づいて公開する権限を持つローカル参加者においてのみ呼び出されます。

次に示すのは実装の例です。

```
const strategy = {

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

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

これは、ユーザーは常に公開状態としたい標準的なビデオチャットアプリケーション用です。オーディオとビデオをミュートまたはミュート解除して、すぐに不可視または可視にできます。(公開/非公開も使用できますが、この方法では大幅に遅くなります。可視性を頻繁に変更したいユースケースには、ミュート/ミュート解除が適しています。)

公開するストリームの選択

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

公開時には、これを使用して公開するオーディオストリームとビデオストリームが決定されます。これについては、後ほど「メディアストリームの公開」で詳しく説明します。

ストラテジーの更新

このストラテジーは動的であることを意図しており、上記の関数のいずれかから返される値はいつでも変更できます。たとえば、ホストアプリケーションがエンドユーザーがボタンをタップするまで公開したくない場合、shouldPublishParticipant (hasUserTappedPublishButton など) から変数を返すことができます。その変数がエンドユーザーの相互作用に基づいて変更されたら、stage.refreshStrategy() を呼び出して、変更されたもののみを適用して、最新の値のストラテジーを照会する必要があることを SDK に通知します。SDK は、shouldPublishParticipant 値が変更されたことを確認すると、公開プロセスを開始します。SDK クエリとすべての関数が以前と同じ値を返す場合、refreshStrategy 呼び出しによってステージが変更されることはありません。

shouldSubscribeToParticipant の戻り値が AUDIO_VIDEO から AUDIO_ONLY に変更され、以前にビデオストリームが存在していた場合は、戻り値が変更されたすべての参加者のビデオストリームが削除されます。

通常、ホストアプリケーションは、適切に管理するために必要なすべての状態について考慮する必要はありません。ステージは以前のストラテジーと現在のストラテジーの違いを最も効率的に適用するストラテジーを使用します。このため、stage.refreshStrategy() の呼び出しはストラテジーが変わらない限り何もしないため、低コストなオペレーションとみなすことができます。

イベント

Stage インスタンスはイベントエミッターです。stage.on() を使用して、ステージの状態がホストアプリケーションに伝達されます。ホストアプリケーションの UI の更新は、通常、イベントによって完全にサポートされます。イベントは次のとおりです。

```
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) => {})
```

これらのイベントのほとんどには、対応する ParticipantInfo が用意されています。

イベントによって提供される情報がストラテジーの戻り値に影響することは想定されていません。たとえば、shouldSubscribeToParticipant の戻り値は、STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED が呼び出されても変化しない想定です。ホストアプリケーションが特定の参加者をサブスクライブする場合は、その参加者の公開状態に関係なく、目的のサブスクリプションタイプを返す必要があります。SDK は、ステージの状態に基づいて、望ましいストラテジーの状態が適切なタイミングで実行されるようにする役目を担います。

メディアストリームを公開する

マイクやカメラなどのローカルデバイスは、上記の「デバイスから を取得する」で説明したのと同じ手順を使用して取得 MediaStream されます。この例では、MediaStream を使用して SDK による公開に使用される LocalStageStream オブジェクトのリストを作成しています。

```
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
    }
}
```

スクリーン共有を公開する

アプリケーションでは、多くの場合、ユーザーの Web カメラに加えてスクリーン共有を公開する必要があります。スクリーン共有を公開するには、独自のトークンを使用して追加の Stage を作成する必要があります。

```
// 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();
```

参加者を表示、削除する

サブスクライブが完了すると、STAGE_PARTICIPANT_STREAMS_ADDED イベントを通じて StageStream オブジェクトの配列を受け取ります。このイベントでは、メディアストリームを表示する際に役立つ参加者情報も提供します。

```
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));
})
```

参加者がストリームの公開を停止したり、配信登録を解除したりすると、削除されたストリームを使用して STAGE_PARTICIPANT_STREAMS_REMOVED 関数が呼び出されます。ホストアプリケーションは、これをシグナルとして使用して、参加者のビデオストリームを DOM から削除する必要があります。

STAGE_PARTICIPANT_STREAMS_REMOVED は、以下を含む、ストリームが削除される可能性のあるすべてのシナリオで呼び出されます。

- リモート参加者は公開を停止します。
- ローカルデバイスがサブスクリプションを解除するか、サブスクリプションを AUDIO_VIDEO から AUDIO_ONLY に変更します。
- リモート参加者がステージを退出します。
- ローカルの参加者がステージを退出します。

STAGE_PARTICIPANT_STREAMS_REMOVED はすべてのシナリオで呼び出されるため、リモートまたはローカルの離脱操作中、 UI から参加者を削除するためのカスタムのビジネスロジックは必要ありません。

メディアストリームをミュート、ミュート解除する

LocalStageStream オブジェクトには、ストリームをミュートするかどうかを制御する setMuted 関数があります。この関数は、stageStreamsToPublish ストラテジー関数から返される前または後にストリームで呼び出すことができます。

重要: refreshStrategy を呼び出した後に新しい LocalStageStream オブジェクトインスタンスが stageStreamsToPublish によって返された場合、新しいストリームオブジェクトのミュート

状態がステージに適用されます。新しい LocalStageStream インスタンスを作成するときは、想定どおりのミュート状態を維持するように注意してください。

リモート参加者のメディアミュート状態の監視

参加者がビデオまたはオーディオのミュート状態を変更すると、変更されたストリームのリストで STAGE_STREAM_MUTE_CHANGED イベントがトリガーされます。StageStream の isMuted プロパティを使用して、次の UI を適宜更新してください。

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

また、オーディオまたはビデオがミュートされているかどうかに関するStageParticipantInfo状態情報も確認できます。

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

WebRTC 統計を取得する

公開中のストリームまたはサブスクライブ中のストリームの最新の WebRTC 統計を取得するには、StageStream の getStats を使用してください。これは、await または promise の連鎖によって統計を取得できる非同期メソッドです。その結果、すべての標準統計を含むディクショナリである RTCStatsReport ができあがります。

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

メディアの最適化

最高のパフォーマンスを得るには、getUserMedia および getDisplayMedia を制限して呼び出すことをお勧めします。

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

次の LocalStageStream コンストラクターに渡される追加オプションを使用して、メディアをさらに制限できます。

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

上記のコードについて

- minBitrate はブラウザが使用することが予想される最小ビットレートを設定します。ただし、ビデオストリームの複雑度が低いと、エンコーダがこのビットレートよりも低くなる可能性があります。
- maxBitrate はこのストリームでブラウザが超えないと予想される最大ビットレートを設定します。
- maxFramerate はこのストリームでブラウザが超えないと予想される最大フレームレートを設定します。
- simulcast オプションは Chromium ベースのブラウザでのみ使用できます。これにより、ストリームの 3 つのレンディションレイヤーを送信できます。
  - これにより、サーバーは、ネットワークの制限に基づいて、他の参加者に送信するレンディションを選択できます。
  - simulcast が maxBitrate および/または maxFramerate の値とともに指定された場合、maxBitrate が内部 SDK の 2 番目に高レイヤーのデフォルト maxBitrate 値である 900

kbps を下回らない限り、これらの値を念頭に置いて最上位のレンディションレイヤーが設定されることが想定されています。

- maxBitrate が 2 番目に高いレイヤーのデフォルト値と比較して低すぎるように指定された場合は、simulcast は無効になります。
- simulcast のオンとオフを切り替えるには、shouldPublishParticipant が false を返し、refreshStrategy を呼び出し、shouldPublishParticipant が true を返し、refreshStrategy をもう一度呼ぶ組み合わせにより、メディアを再パブリッシュする必要があります。

参加者属性を取得

CreateParticipantToken エンドポイントリクエストで属性を指定すると、次のように StageParticipantInfo プロパティにその属性が表示されます。

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

ネットワーク問題の処理

ローカルデバイスのネットワーク接続が失われると、SDK はユーザーアクションなしで内部で再接続を試みます。場合によっては、SDK が正常に動作せず、ユーザーアクションが必要なる可能性があります。

大まかに、ステージの状態は STAGE_CONNECTION_STATE_CHANGED イベントを介して処理できます。

```
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;
})
```

一般に、ステージに正常に参加した後にエラーが発生した場合、SDK が接続を失い、接続を再確立できなかったことを示します。新しい Stage オブジェクトを作成し、ネットワークの状態が向上したら参加してみます。

IVS チャネルにステージをブロードキャストする

ステージをブロードキャストするには、IVSBroadcastClient セッションを作成してから、前述の SDK による通常のブロードキャスト手順に従います。次のように、STAGE_PARTICIPANT_STREAMS_ADDED を介して公開された StageStream のリストを使用して、ブロードキャスト ストリーム構成に適用できる参加者メディア ストリームを取得できます。

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

オプションで、ステージを合成して IVS 低レイテンシーチャネルにブロードキャストすることで、より多くの視聴者に届けることもできます。「IVS 低レイテンシーストリーミングユーザーガイド」の「Amazon IVS ストリームでの複数のホストの有効化」を参照してください。

既知の問題と回避策

- `stage.leave()` を呼び出さずにブラウザのタブを閉じたり、ブラウザを終了したりしても、ユーザーは最大 10 秒間フレームがフリーズしたり、画面が真っ暗になったりしてセッションに表示されることがあります。

  回避策: 該当なし。

- セッションの開始後、Safari セッションに参加しているユーザーには、断続的に黒い画面が表示されます。

  回避策: ブラウザを更新して、セッションに再接続します。

- Safari は、ネットワークを切り替えても正常に回復しません。

  回避策: ブラウザを更新して、セッションに再接続します。

- 開発者コンソールは `Error: UnintentionalError at StageSocket.onClose` エラーを繰り返します。

  回避策: 参加者トークンごとに作成できるステージは 1 つだけです。このエラーは、`Stage` インスタンスが 1 つのデバイス上にあるか複数のデバイスにあるかに関係なく、同じ参加者トークンで複数のインスタンスが作成された場合に発生します。

- `StageParticipantPublishState.PUBLISHED` 状態の維持に問題がある場合や、`StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`イベントをリッスンするときに`StageParticipantPublishState.ATTEMPTING_PUBLISH`状態が繰り返されることがあります。

  回避策： `getUserMedia`または を呼び出すときに、動画解像度を 720p に制限しますgetDisplayMedia。具体的には、幅`getUserMedia`と高さの および `getDisplayMedia`制約値を乗算すると、921600 (1280*720) を超えないようにしてください。

Safari での制限事項

- アクセス許可のプロンプトを拒否するには、Safari でのウェブサイト設定のアクセス許可を OS レベルでリセットする必要があります。
- Safari は、Firefox や Chrome ほど効果的にすべてのデバイスをネイティブに検出されません。例えば、OBS 仮想カメラは検知されません。

Firefox での制限事項

- Firefox で画面を共有するには、システムのアクセス許可を有効にする必要があります。有効にした後、正常に動作するには Firefox を再起動する必要があります。再起動しないと、アクセス許可がブロックされていると認識されると、ブラウザは NotFoundError 例外をスローします。
- `getCapabilities` メソッドがありません。これは、ユーザーがメディアトラックの解像度やアスペクト比を取得できないことを意味します。こちらの Bugzilla のスレッドを参照してください。
- レイテンシーやチャネル数などの、いくつかの `AudioContext` プロパティがありません。これは、オーディオトラックを操作する上級ユーザーにとって問題になる可能性があります。

- `getUserMedia` からのカメラフィードは、MacOS でのアスペクト比が 4:3 に制限されています。Bugzilla のスレッド 1 および Bugzilla のスレッド 2 を参照してください。

- オーディオキャプチャが `getDisplayMedia` でサポートされていません。こちらの Bugzilla のスレッドを参照してください。
- スクリーンキャプチャのフレームレートが最適ではありません (約 15 fps?)。こちらの Bugzilla のスレッドを参照してください。

モバイルウェブの制限事項

- getDisplayMedia 画面共有はモバイルデバイスではサポートされていません。

  回避策: 該当なし。
- `leave()` を呼び出さずにブラウザを閉じると、参加者が退出するまでに 15～30 秒かかります。

  回避策: ユーザーが正しく接続を解除するように促す UI を追加します。
- バックグラウンドで動くアプリケーションが原因で動画の公開が停止します。

  回避策: パブリッシャーが一時停止しているときに UI スレートを表示します。
- Android デバイスでカメラのミュートを解除すると、動画のフレームレートが約 5 秒間低下します。

  回避策: 該当なし。
- iOS 16.0 では、ビデオフィードはローテーション時にストレッチされます。

回避策: OS の既知の問題の概要を示す UI を表示します。

- オーディオ入力デバイスを切り替えると、オーディオ出力デバイスも自動的に切り替わります。

  回避策: 該当なし。

- ブラウザを背景に設定すると、公開ストリームが黒になり、オーディオのみが生成されます。

  回避策: 該当なし。これはセキュリティ上の理由です。

エラー処理

このセクションでは、エラー状態の概要、Web Broadcast SDK がエラー状態をアプリケーションに報告する方法、およびエラー発生時にアプリケーションが実行すべき処理について説明します。エラーには次の 4 つのカテゴリがあります。

```
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
  }
});
```

Stage インスタンス化エラー

Stage インスタンス化では、トークンがリモートで検証されませんが、クライアント側で検証可能な基本的なトークンの問題がないかチェックされます。その結果、SDK がエラーをスローする場合があります。

不正な形式の参加者トークン

これは、ステージトークンの形式が不正な場合に発生します。Stage をインスタンス化すると、SDK は「Error parsing Stage Token」というエラーメッセージをスローします。

アクション: 有効なトークンを作成し、インスタンス化を再試行してください。

Stage 参加エラー

これは、初めてステージに参加しようとしたときに発生する場合がある工ラーです。

Stage が削除されました

これは、削除された (トークンに関連付けられた) ステージに参加したときに発生します。`join` SDK メソッドは、「10 秒InitialConnectTimedOut 後」というエラーメッセージをスローします。

アクション: 新しいステージで有効なトークンを作成し、参加を再試行してください。

期限切れの参加者トークン

これはトークンの有効期限が切れた場合に発生します。`join` SDK メソッドが「Token expired and is no longer valid」というエラーメッセージをスローします。

アクション: 新しいトークンを作成し、参加を再試行してください。

無効または取り消された参加者トークン

これは、トークンが有効でないか、取り消された/切断された場合に発生します。`join` SDK メソッドは、「10 秒InitialConnectTimedOut 後」というエラーメッセージをスローします。

アクション: 新しいトークンを作成し、参加を再試行してください。

切断されたトークン

これは、ステージトークンの形式が不正でないにもかかわらず、Stages サーバーによって拒否された場合に発生します。`join` SDK メソッドは、「10 秒InitialConnectTimedOut 後」というエラーメッセージをスローします。

アクション: 有効なトークンを作成し、参加を再試行してください。

初期参加におけるネットワークエラー

これは、SDK が Stages サーバーにアクセスしたにもかかわらず、接続を確立できなかった場合に発生します。join SDK メソッドは、「10 秒InitialConnectTimedOut 後」というエラーメッセージをスローします。

アクション: デバイスの接続が回復するのを待ってから、参加を再試行してください。

既に参加している場合のネットワークエラー

デバイスのネットワーク接続が切断されると、SDK と Stages サーバーとの接続が失われる可能性があります。SDK がバックエンドサービスにアクセスできなくなるため、コンソールにエラーが表示される場合があります。https://broadcast.stats.live-video.net への POST は失敗します。

公開中/サブスクライブ中の場合は、公開/サブスクライブの試行に関連するエラーがコンソールに表示されます。

SDK は、エクスポネンシャルバックオフストラテジーを使用して内部で再接続を試みます。

アクション: デバイスの接続が回復するまでお待ちください。公開中またはサブスクライブ中の場合は、ストラテジーを更新してメディアストリームを確実に再公開できるようにします。

公開エラーとサブスクライブエラー

公開エラー: 公開状態

SDK は公開が失敗したときに ERRORED を報告します。これは、ネットワークの状態が原因で発生する可能性がありますが、ステージがパブリッシャーのキャパシティに達した場合にも発生することがあります。

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

アクション: ストラテジーを更新して、メディアストリームの再公開を試みてください。

サブスクライブエラー

SDK はサブスクライブが失敗したときに ERRORED を報告します。これは、ネットワークの状態が原因で発生する可能性がありますが、ステージがサブスクライバーのキャパシティに達した場合にも発生することがあります。

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

アクション: ストラテジーを更新して、新しいサブスクライブを試してください。

IVS Broadcast SDK: Android ガイド (リアルタイムストリーミング)

IVS リアルタイムストリーミングの Android Broadcast SDK を使用して、参加者は Android 上でビデオを送受信できます。

`com.amazonaws.ivs.broadcast` パッケージは、このドキュメントで説明されているインターフェイスを実装します。SDK は、次の操作をサポートします。

- ステージに参加する
- ステージ内の他の参加者にメディアを公開する
- ステージ内の他の参加者のメディアをサブスクライブする
- ステージに公開されたビデオとオーディオを管理および監視する
- 各ピア接続の WebRTC 統計を取得
- IVS 低レイテンシーストリーミング Android Broadcast SDK からのすべての操作

Android Broadcast SDK の最新バージョン： 1.14.1 (リリースノート)

リファレンスドキュメント： Amazon IVS Android Broadcast SDK で使用できる最も重要な方法については、https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/android/ のリファレンスドキュメントを参照してください。

サンプルコード: GitHubhttps://github.com/aws-samples/amazon-ivs-broadcast-android-sample の Android サンプルリポジトリを参照してください。

プラットフォーム要件: Android 9.0 以降。

開始方法

ライブラリのインストール

Amazon IVS Android Broadcast ライブラリを Android 開発環境に追加するには、ライブラリを以下に示すモジュールの `build.gradle` ファイル (最新バージョンの Amazon IVS Broadcast SDK) に追加します。

```
repositories {
    mavenCentral()
}

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

マニフェストに次の権限を追加して、SDK がスピーカーフォンを有効または無効にできるようにします。

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

または、SDK を手動でインストールするには、次の場所から最新バージョンをダウンロードします。

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

必ず `aar` に `-stages` が付与されたものをダウンロードしてください。

必要なアクセス許可

アプリはユーザーのカメラとマイクへのアクセス許可を要求する必要があります。(これは、Amazon IVS に特有なものではなく、カメラやマイクにアクセスする必要があるアプリケーションには必須です。)

ここでは、ユーザーがすでにアクセス許可を付与しているかどうかを確認し、付与していない場合は、許可を求めます。

```
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;
    }
}
```

ここでは、ユーザーの応答を取得します。

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

公開とサブスクライブ

概念

リアルタイム機能には、ステージ、ストラテジー、レンダラーという 3 つのコアコンセプトがあります。設計目標は、実際に動作する製品を構築するのに必要となるクライアント側ロジックの量を最小限に抑えることです。

ステージ

Stage クラスは、ホストアプリケーションと SDK 間の主要な相互作用のポイントです。これはステージそのものを表し、ステージへの参加とステージからの退出に使用されます。ステージの作成と

参加には、コントロールプレーンからの有効で有効期限内のトークン文字列 (token として表示) が必要です。ステージへの参加と退出は簡単です。

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

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

stage.leave();
```

また、Stage クラスには StageRenderer をアタッチすることもできます。

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

方針

Stage.Strategy インターフェースは、ホストアプリケーションがステージの望ましい状態を SDK に伝える方法を提供しま
す。shouldSubscribeToParticipant、shouldPublishFromParticipant、stageStreamsToPubl
の 3 つの関数を実装する必要があります。以下で、すべて説明します。

参加者へのサブスクライブ

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

リモート参加者がステージに参加すると、SDK はその参加者に対して希望するサブスクリプションの状態についてホストアプリケーションに問い合わせます。使用できるオプションは NONE、AUDIO_ONLY、および AUDIO_VIDEO です。この関数の値を返す場合、ホストアプリケーションは公開の状態、現在のサブスクリプションの状態、またはステージ接続の状態を考慮する必要はありません。AUDIO_VIDEO が返された場合、SDK はリモート参加者が公開するまで待ってからサブスクライブし、プロセス全体でレンダラーを通じてホストアプリケーションを更新します。

次に示すのは実装の例です。

```
@Override
```

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

これは、ビデオチャットアプリケーションなど、すべての参加者が互いに常に可視化されているホストアプリケーション向けのの完全な実装です。

より高度な実装も可能です。ParticipantInfo の userInfo プロパティを使用して、サーバーが提供する属性に基づいて、参加者に対して選択的にサブスクライブできます。

```
@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;
 }
}
```

これを使用すると、モデレーターは、自身は視聴の対象とならずに、すべてのゲストを監視できるステージを作ることができます。ホストアプリケーションでは、追加のビジネスロジックを使用して、モデレーター同士は見えるようにしつつ、ゲストには見えないようにすることができます。

公開

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

ステージに接続すると、SDK はホストアプリケーションにクエリを実行し、特定の参加者を公開とすべきかどうかを確認します。これは、提供されたトークンに基づいて公開する権限を持つローカル参加者においてのみ呼び出されます。

次に示すのは実装の例です。

```
@Override
```

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

これは、ユーザーは常に公開状態としたい標準的なビデオチャットアプリケーション用です。オーディオとビデオをミュートまたはミュート解除して、すぐに不可視または可視にできます。(公開/非公開も使用できますが、この方法では大幅に遅くなります。可視性を頻繁に変更したいユースケースには、ミュート/ミュート解除が適しています。)

公開するストリームの選択

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

公開時には、これを使用して公開するオーディオストリームとビデオストリームが決定されます。これについては、後ほど「メディアストリームの公開」で詳しく説明します。

ストラテジーの更新

このストラテジーは動的であることを意図しており、上記の関数のいずれかから返される値はいつでも変更できます。たとえば、ホストアプリケーションがエンドユーザーがボタンをタップするまで公開したくない場合、shouldPublishFromParticipant (hasUserTappedPublishButton など) から変数を返すことができます。その変数がエンドユーザーの相互作用に基づいて変更されたら、stage.refreshStrategy() を呼び出して、変更されたもののみを適用して、最新の値のストラテジーを照会する必要があることを SDK に通知します。SDK は、shouldPublishFromParticipant 値が変更されたことを検出すると、公開プロセスを開始します。SDK クエリとすべての関数が以前と同じ値を返す場合、refreshStrategy 呼び出しはステージに変更を加えません。

shouldSubscribeToParticipant の戻り値が AUDIO_VIDEO から AUDIO_ONLY に変更され、以前にビデオストリームが存在していた場合は、戻り値が変更されたすべての参加者のビデオストリームが削除されます。

通常、ホストアプリケーションは、適切に管理するために必要なすべての状態について考慮する必要はありません。ステージは以前のストラテジーと現在のストラテジーの違いを最も効率的に適用するストラテジーを使用します。このため、stage.refreshStrategy() の呼び出しはストラテジーが変わらない限り何もしないため、低コストなオペレーションとみなすことができます。

レンダラー

StageRenderer インターフェースはステージの状態をホストアプリケーションに伝えます。ホストアプリケーションの UI の更新は、通常、レンダラーが提供するイベントだけで行うことができます。次の関数では、以下のような結果が生成されます。

```
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);
```

これらのメソッドのほとんどには、対応する Stage および ParticipantInfo が用意されています。

レンダラーから提供された情報がストラテジーの戻り値に影響することは想定されていません。たとえば、shouldSubscribeToParticipant の戻り値は、onParticipantPublishStateChanged が呼び出されても変化しない想定です。ホストアプリケーションが特定の参加者をサブスクライブする場合は、その参加者の公開状態に関係なく、目的のサブスクリプションタイプを返す必要があります。SDK は、ステージの状態に基づいて、望ましいストラテジーの状態が適切なタイミングで実行されるようにする役目を担います。

StageRenderer は次のステージクラスにアタッチできます。

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

公開参加者のみが onParticipantJoined をトリガーし、参加者が公開を停止するか、ステージセッションを終了すると、onParticipantLeft がトリガーされることに注意してください。

メディアストリームを公開する

内蔵マイクやカメラなどのローカルデバイスは、DeviceDiscovery を介して検出されます。以下は、前面カメラとデフォルトのマイクを選択し、それらを LocalStageStreams として SDK が公開できるように返す例です。

```
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;
```

```
}
```

参加者を表示、削除する

サブスクライブが完了すると、レンダラーの StageStream 関数を介して onStreamsAdded オブジェクトの配列を受け取ります。ImageStageStream からのプレビューは次の場所から取得できます。

```
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);
```

オーディオレベルの統計情報は、以下の AudioStageStream から取得できます。

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

参加者が公開を停止するか、サブスクライブを解除すると、削除されたストリームを使用して onStreamsRemoved 関数が呼び出されます。ホストアプリケーションは、これを通知として使用して、参加者のビデオストリームをビュー階層から削除する必要があります。

onStreamsRemoved は、以下を含む、ストリームが削除される可能性のあるすべてのシナリオで呼び出されます。

- リモート参加者は公開を停止します。
- ローカルデバイスがサブスクリプションを解除するか、サブスクリプションを AUDIO_VIDEO から AUDIO_ONLY に変更します。
- リモート参加者がステージを退出します。
- ローカルの参加者がステージを退出します。

onStreamsRemoved はすべてのシナリオで呼び出されるため、リモートまたはローカルの離脱操作中、 UI から参加者を削除するためのカスタムのビジネスロジックは必要ありません。

メディアストリームをミュート、ミュート解除する

LocalStageStream オブジェクトには、ストリームをミュートするかどうかを制御する setMuted 関数があります。この関数は、streamsToPublishForParticipant ストラテジー関数から返される前または後にストリームで呼び出すことができます。

重要: refreshStrategy を呼び出した後に新しい LocalStageStream オブジェクトインスタンスが streamsToPublishForParticipant によって返された場合、新しいストリームオブジェクトのミュート状態がステージに適用されます。新しい LocalStageStream インスタンスを作成するときは、想定どおりのミュート状態を維持するように注意してください。

リモート参加者のメディアミュート状態の監視

参加者がビデオまたはオーディオストリームのミュート状態を変更すると、変更されたストリームのリストとともにレンダラー onStreamMutedChanged 関数が呼び出されます。StageStream に getMuted メソッドを使用して、適宜 UI を更新します。

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

WebRTC 統計を取得する

公開ストリームまたはサブスクライブ中のストリームの最新の WebRTC 統計情報を取得するには、StageStream に requestRTCStats を使用してください。収集が完了すると、StageStream に設定できる StageStream.Listener から統計を受け取ります。

```
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());
```

```
  }
 }
}
```

参加者属性を取得

CreateParticipantToken エンドポイントリクエストで属性を指定すると、次のように ParticipantInfo プロパティにその属性が表示されます。

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

セッションをバックグラウンドで続行

アプリがバックグラウンドに入ると、公開を停止するか、他のリモート参加者の音声のみをサブスクライブすることができます。そのためには、Strategy 実装を更新して公開を停止し、AUDIO_ONLY (または NONE。該当する場合) をサブスクライブします。

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

サイマルキャストによるレイヤードエンコーディングの有効化/無効化

メディアストリームを公開する際、SDK は高品質と低品質のビデオストリームを伝送するため、リモートの参加者はダウンリンクの帯域幅が限られていてもストリームをサブスクライブできます。サイマルキャストによるレイヤードエンコーディングはデフォルトでオンになっています。次の StageVideoConfiguration.Simulcast クラスを使用すると無効にできます。

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

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

ビデオ設定の制限

SDK は、StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size) を使用したポートレートモードまたはランドスケープモードの強制をサポートしていません。縦の向きでは、小さい方の寸法が幅として使用され、横の向きでは高さとして使用されます。つまり、次の 2 つの setSize への呼び出しが動画の設定に同じ効果をもたらすということです。

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

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

ネットワーク問題の処理

ローカルデバイスのネットワーク接続が失われると、SDK はユーザーアクションなしで内部で再接続を試みます。場合によっては、SDK が正常に動作せず、ユーザーアクションが必要なる可能性があります。ネットワーク接続の切断に関連する主なエラーは 2 つあります。

- エラーコード 1400、メッセージ：PeerConnection 「不明なネットワークエラーにより失われました」
- エラーコード 1300、メッセージ:「再試行の試行回数制限に達しました」

最初のエラーを受け取って 2 番目のエラーを受け取らない場合、SDK はまだステージに接続されており、自動的に接続を再確立しようとします。安全対策として、ストラテジーメソッドの戻り値を変更せずに `refreshStrategy` を呼び出すと、手動で再接続を試みることができます。

2 番目のエラーを受け取った場合、SDK の再接続は失敗し、ローカルデバイスはステージに接続されていません。この場合は、ネットワーク接続が再確立された後に `join` を呼び出してステージに再び参加してみてください。

一般に、ステージに正常に参加した後にエラーが発生した場合は、SDK が接続を再確立できなかったことを示します。新しい `Stage` オブジェクトを作成し、ネットワークの状態が向上したら参加してみます。

Bluetooth マイクの使用

Bluetooth マイクデバイスを使用して公開するには、Bluetooth SCO 接続を開始する必要があります。

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

既知の問題と回避策

- Android デバイスがスリープ状態になってから起動すると、プレビューがフリーズ状態になる場合があります。

  回避策: 新しい `Stage` を作成して使用します。
- 参加者が別の参加者が使用しているトークンを使用して参加すると、最初の接続は特別なエラーなしで切断されます。

  回避策: 該当なし。
- まれにパブリッシャーが公開中でも、サブスクライバーが受け取る公開状態が `inactive` であるという問題が発生します。

回避策:セッションを終了してから再度参加してみてください。問題が解決しない場合は、パブリッシャー用に新しいトークンを作成してください。

- ステージセッション中、通常は通話時間が長くなると、まれにオーディオディストーションの問題が断続的に発生することがあります。

  回避策: オーディオディストーションの問題が起きる参加者は、セッションを終了してから再度参加するか、音声を非公開にしてから再公開して問題を解決できます。
- ステージに公開する場合、外部マイクはサポートされていません。

  回避策: USB で接続された外部マイクをステージへの公開に使用しないでください。
- `createSystemCaptureSources` を使用して画面共有を使用したステージへの公開はサポートされていません。

  回避策: カスタムの画像入力ソースとカスタムオーディオ入力ソースを使用して、システムキャプチャを手動で管理します。
- `ImagePreviewView` が親から削除された場合 (例えば、`removeView()` が親側で呼び出された場合)、`ImagePreviewView` はすぐにリリースされます。`ImagePreviewView` を別の親ビューに追加しても、フレームは表示されません。

  回避策: `getPreview` を使用して別のプレビューをリクエストします。
- Android 12 搭載の Samsung Galaxy S22/+ でステージに参加すると、1401 エラーが発生し、ローカルデバイスがステージに参加できない、または参加しても音声が聞こえない場合があります。

  回避策: Android 13 にアップグレードします。
- Android 13 搭載の Nokia X20 でステージに参加すると、カメラが開かず、例外が発生する場合があります。

  回避策: 該当なし。
- MediaTek Helio チップセットを搭載したデバイスでは、リモート参加者のビデオが正しくレンダリングされない場合があります。

  回避策: 該当なし。
- 一部のデバイスでは、デバイスの OS が SDK で選択したものとは異なるマイクを選択する場合があります。これは、`VOICE_COMMUNICATION` オーディオルートの定義方法はデバイスメーカーによって異なるため、Amazon IVS Broadcast SDK では制御できないためです。

  回避策: 該当なし。

- 一部の Android ビデオエンコーダーは、ビデオサイズが 176 x 176 未満の場合は設定できません。サイズを小さく設定するとエラーが発生し、ストリーミングが妨げられます。

  回避策： ビデオサイズを 176 x 176 未満に設定しないでください。

エラー処理

致命的なエラーと致命的でないエラー

エラーオブジェクトには、`BroadcastException` の「is fatal」ブール値フィールドが含まれています。

一般的に、致命的なエラーは Stages サーバーへの接続に関連しています (接続を確立できないか、接続が失われて回復できないかのいずれか)。アプリケーションはステージを再作成し、(場合によっては新しいトークンを使って、あるいはデバイスの接続が回復したときに) 再参加する必要があります。

致命的でないエラーは通常、公開/サブスクライブ状態に関連しており、公開/サブスクライブ操作を再試行する SDK によって処理されます。

次のプロパティを確認できます。

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

参加エラー

不正な形式のトークン

これは、ステージトークンの形式が不正な場合に発生します。

SDK は、`stage.join` への呼び出しで Java 例外 (error code = 1000 および fatal = true) をスローします。

アクション: 有効なトークンを作成し、参加を再試行してください。

期限切れのトークン

これは、ステージトークンの有効期限が切れた場合に発生します。

SDK は、`stage.join` への呼び出しで Java 例外 (error code = 1001 および fatal = true) をスローします。

アクション: 新しいトークンを作成し、参加を再試行してください。

無効または取り消されたトークン

これは、ステージトークンの形式が不正でないにもかかわらず、Stages サーバーによって拒否された場合に発生します。これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK が `onConnectionStateChanged` を呼び出すと、error code = 1026 および fatal = true の例外が発生します。

アクション: 有効なトークンを作成し、参加を再試行してください。

初期参加におけるネットワークエラー

これは、SDK が Stages サーバーにアクセスしたにもかかわらず、接続を確立できなかった場合に発生します。これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK が `onConnectionStateChanged` を呼び出すと、error code = 1300 および fatal = true の例外が発生します。

アクション: デバイスの接続が回復するのを待ってから、参加を再試行してください。

既に参加している場合のネットワークエラー

デバイスのネットワーク接続が切断されると、SDK と Stages サーバーとの接続が失われる可能性があります。これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK が `onConnectionStateChanged` を呼び出すと、error code = 1300 および fatal = true の例外が発生します。

アクション: デバイスの接続が回復するのを待ってから、参加を再試行してください。

公開/サブスクライブエラー

初期

次のようなエラーがあります。

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

これらは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK は限られた回数だけ操作を再試行します。再試行中の公開/サブスクライブ状態は ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE です。再試行が成功すると、状態は PUBLISHED/SUBSCRIBED に変更されます。

SDK が onError を呼び出すと、関連するエラーコードおよび fatal = true が発生します。

アクション: SDK は自動的に再試行するため、アクションは不要です。オプションで、アプリケーションはストラテジーを更新して再試行回数を増やすことができます。

確立済みにもかかわらず失敗する

公開またはサブスクライブは、確立された後に失敗することがあります (ネットワークエラーが原因である可能性が高い)。「ネットワークエラーによりピア接続が失われた」ことを示すエラーコードは 1400 です。

これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK は、公開/サブスクライブ操作を再試行します。再試行中の公開/サブスクライブ状態は ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE です。再試行が成功すると、状態は PUBLISHED/SUBSCRIBED に変更されます。

SDK が onError を呼び出すと、error code = 1400 および fatal = false が発生します。

アクション: SDK は自動的に再試行するため、アクションは不要です。オプションで、アプリケーションはストラテジーを更新して再試行回数を増やすことができます。接続が完全に失われた場合、Stages への接続も失敗する可能性があります。

IVS Broadcast SDK: iOS ガイド (リアルタイムストリーミング)

IVS リアルタイムストリーミング iOS Broadcast SDK を使用すると、参加者は iOS でビデオを送受信できます。

`AmazonIVSBroadcast` モジュールは、このドキュメントで説明されているインターフェイスを実装します。以下のオペレーションがサポートされています。

- ステージに参加する
- ステージ内の他の参加者にメディアを公開する
- ステージ内の他の参加者のメディアをサブスクライブする
- ステージに公開されたビデオとオーディオを管理および監視する
- 各ピア接続の WebRTC 統計を取得
- IVS 低レイテンシーストリーミング iOS Broadcast SDK からのすべての操作

iOS Broadcast SDK の最新バージョン： 1.14.1 (リリースノート)

リファレンスドキュメント： Amazon IVS iOS Broadcast SDK で使用できる最も重要な方法については、https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/ios/ のリファレンスドキュメントを参照してください。

サンプルコード: : GitHubhttps://github.com/aws-samples/amazon-ivs-broadcast-ios-sample の iOS サンプルリポジトリを参照してください。

プラットフォームの要件: iOS 14 以上です。

開始方法

ライブラリのインストール

Broadcast SDK は 経由で統合することをお勧めします CocoaPods。(代わりに、フレームワークを手動でプロジェクトに追加することも可能です)。

推奨: Broadcast SDK の統合 (CocoaPods)

リアルタイム機能は iOS 低レイテンシーストリーミング Broadcast SDK のサブスペックとして公開されています。これは、お客様が機能のニーズに基づいて含めるか除外するかを選択できるようにするためです。含めるとパッケージサイズが大きくなります。

リリースは、 という名前 CocoaPods の で公開されますAmazonIVSBroadcast。この依存関係を自分の Podfile に追加します。

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

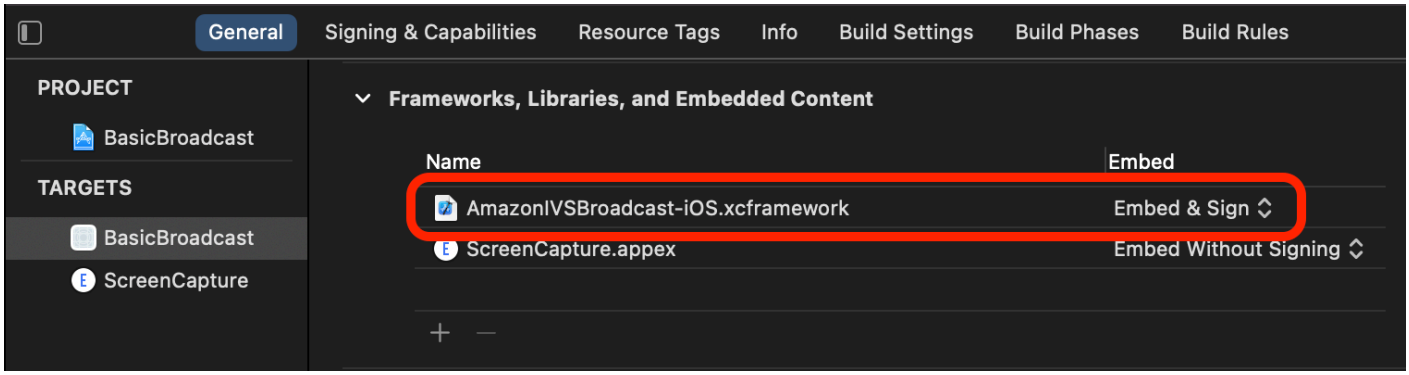
pod install を実行すると、SDK が .xcworkspace で利用できるようになります。

重要: IVS リアルタイムストリーミング Broadcast SDK (ステージサブスペック付き) には、IVS 低レイテンシーストリーミング Broadcast SDK のすべての機能が含まれています。両方の SDK を同じプロジェクトに統合することはできません。を介してステージサブ仕様 CocoaPods をプロジェクトに追加する場合は、 を含むポッドファイル内の他の行をすべて削除してくださいAmazonIVSBroadcast。たとえば、Podfile には次の行を両方含めないでください。

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

代替方法: フレームワークを手動でインストールする

1. https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip から最新バージョンをダウンロードします。
2. アーカイブの内容を抽出します。AmazonIVSBroadcast.xcframework には、デバイスとシミュレータの両方の SDK が含まれています。
3. アプリケーションターゲットの [全般] タブの、[Frameworks, Libraries, and Embedded Content (フレームワーク、ライブラリ、埋め込みコンテンツ)] のセクションに AmazonIVSBroadcast.xcframework をドラッグして埋め込みます。



必要なアクセス許可

アプリはユーザーのカメラとマイクへのアクセス許可を要求する必要があります。(これは、Amazon IVS に特有なものではなく、カメラやマイクにアクセスする必要があるアプリケーションには必須です。)

ここでは、ユーザーがすでにアクセス許可を付与しているかどうかを確認し、付与していない場合は、許可を求めます。

```
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.
}
```

カメラやマイクにアクセスするには、.video と .audio の両方のメディアタイプに対してこれを行う必要があります。

また、NSCameraUsageDescription と NSMicrophoneUsageDescription のエントリを Info.plist に追加する必要があります。これを行わずにアクセス許可をリクエストすると、アプリがクラッシュします。

アプリケーションアイドルタイマーの無効化

これはオプションですが推奨されます。Broadcast SDK の使用中にデバイスがスリープ状態になり、ブロードキャストが中断されるのを防ぎます。

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

公開とサブスクライブ

概念

リアルタイム機能には、ステージ、ストラテジー、レンダラーという 3 つのコアコンセプトがあります。設計目標は、実際に動作する製品を構築するのに必要となるクライアント側ロジックの量を最小限に抑えることです。

ステージ

IVSStage クラスは、ホストアプリケーションと SDK 間の主要な相互作用のポイントです。クラスはステージそのものを表し、ステージへの参加とステージからの退出に使用されます。ステージを作成または参加するには、コントロールプレーンからの有効期限が切れていない有効なトークン文字列 (token として表されます) が必要です。ステージへの参加と退出は簡単です。

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

try stage.join()

stage.leave()
```

また、IVSStage クラスには次の IVSStageRenderer および IVSErrorDelegate をアタッチすることもできます。

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

方針

IVSStageStrategy プロトコルは、ホストアプリケーションがステージの望ましい状態を SDK に伝達する方法を提供します。shouldSubscribeToParticipant、shouldPublishParticipant、streamsToPublishForPa の 3 つの関数を実装する必要があります。以下で、すべて説明します。

参加者へのサブスクライブ

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

リモート参加者がステージに参加すると、SDK はその参加者に対して希望するサブスクリプションの状態についてホストアプリケーションに問い合わせます。使用できるオプションは .none、.audioOnly、および .audioVideo です。この関数の値を返す場合、ホストアプリケーションは公開の状態、現在のサブスクリプションの状態、またはステージ接続の状態を考慮する必要はありません。.audioVideo が返された場合、SDK はリモート参加者が公開するまで待ってからサブスクライブし、プロセス全体でレンダラーを通じてホストアプリケーションを更新します。

次に示すのは実装の例です。

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

これは、ビデオチャットアプリケーションなど、すべての参加者がお互いに常に会えるホストアプリケーション向けのこの機能の完全な実装です。

より高度な実装も可能です。IVSParticipantInfo の attributes プロパティを使用して、サーバーが提供する属性に基づいて、参加者に対して選択的にサブスクライブできます。

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

これを使用すると、モデレーターは、自身は視聴の対象とならずに、すべてのゲストを監視できるステージを作ることができます。ホストアプリケーションでは、追加のビジネスロジックを使用して、モデレータがお互いを見えるようにしても、ゲストには見えないようにすることができます。

公開

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

ステージに接続すると、SDK はホストアプリケーションにクエリを実行し、特定の参加者を公開とすべきかどうかを確認します。これは、提供されたトークンに基づいて公開する権限を持つローカル参加者においてのみ呼び出されます。

次に示すのは実装の例です。

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

これは、ユーザーは常に公開状態としたい標準的なビデオチャットアプリケーション用です。オーディオとビデオをミュートまたはミュート解除して、すぐに不可視または可視にできます。(公開/非公開も使用できますが、この方法では大幅に遅くなります。可視性を頻繁に変更したいユースケースには、ミュート/ミュート解除が適しています。)

公開するストリームの選択

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

公開時には、これを使用して公開するオーディオストリームとビデオストリームが決定されます。これについては、後ほど「メディアストリームの公開」で詳しく説明します。

ストラテジーの更新

このストラテジーは動的であることを意図しており、上記の関数のいずれかから返される値はいつでも変更できます。たとえば、ホストアプリケーションがエンドユーザーがボタンをタップするまで公開したくない場合、shouldPublishParticipant (hasUserTappedPublishButton など) から変数を返すことができます。その変数がエンドユーザーの相互作用に基づいて変更されたら、stage.refreshStrategy() を呼び出して、変更されたもののみを適用して、最新の値のストラテジーを照会する必要があることを SDK に通知します。SDK は、shouldPublishParticipant 値が変更されたことを検出すると、公開プロセスを開始します。SDK クエリとすべての関数が以前と同じ値を返す場合、refreshStrategy 呼び出しによってステージが変更されることはありません。

shouldSubscribeToParticipant の戻り値が .audioVideo から .audioOnly に変更され、以前にビデオストリームが存在していた場合は、戻り値が変更されたすべての参加者のビデオストリームが削除されます。

通常、ホストアプリケーションは、適切に管理するために必要なすべての状態について考慮する必要はありません。ステージは以前のストラテジーと現在のストラテジーの違いを最も効率的に適用するストラテジーを使用します。このため、stage.refreshStrategy() の呼び出しはストラテジーが変わらない限り何もしないため、低コストなオペレーションとみなすことができます。

レンダラー

IVSStageRenderer プロトコルはステージの状態をホストアプリケーションに伝えます。ホストアプリケーションの UI の更新は、通常、レンダラーが提供するイベントだけで行うことができます。次の関数では、以下のような結果が生成されます。

```
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?)
```

レンダラーから提供された情報がストラテジーの戻り値に影響することは想定されていません。たとえば、shouldSubscribeToParticipant の戻り値は、participant:didChangePublishState が呼び出されても変化しない想定です。ホストアプリケーションが特定の参加者をサブスクライブする場合は、その参加者の公開状態に関係なく、目的のサブスクリプションタイプを返す必要があります。SDK は、ステージの状態に基づいて、望ましいストラテジーの状態が適切なタイミングで実行されるようにする役目を担います。

公開参加者のみが participantDidJoin をトリガーし、参加者が公開を停止するか、ステージセッションを終了すると、participantDidLeave がトリガーされることに注意してください。

メディアストリームを公開する

内蔵マイクやカメラなどのローカルデバイスは、IVSDeviceDiscovery を介して検出されます。以下は、前面カメラとデフォルトのマイクを選択し、それらを IVSLocalStageStreams として SDK で公開できるように戻す例です。

```
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]
}
```

参加者を表示、削除する

サブスクライブが完了すると、レンダラーの IVSStageStream 関数を介して didAddStreams オブジェクトの配列を受け取ります。この参加者のオーディオレベル統計をプレビューまたは受信するには、ストリームから基になる IVSDevice オブジェクトにアクセスできます。

```
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 */
    })
```

```
}
```

参加者が公開を停止するか、サブスクライブを解除すると、削除されたストリームを使用して didRemoveStreams 関数が呼び出されます。ホストアプリケーションは、これを通知として使用して、参加者のビデオストリームをビュー階層から削除する必要があります。

didRemoveStreams は、以下を含む、ストリームが削除される可能性のあるすべてのシナリオで呼び出されます。

- リモート参加者は公開を停止します。
- ローカルデバイスがサブスクリプションを解除するか、サブスクリプションを .audioVideo から .audioOnly に変更します。
- リモート参加者がステージを退出します。
- ローカルの参加者がステージを退出します。

didRemoveStreams はすべてのシナリオで呼び出されるため、リモートまたはローカルの離脱操作中、 UI から参加者を削除するためのカスタムのビジネスロジックは必要ありません。

メディアストリームをミュート、ミュート解除する

IVSLocalStageStream オブジェクトには、ストリームをミュートするかどうかを制御する setMuted 関数があります。この関数は、streamsToPublishForParticipant ストラテジー関数から返される前または後にストリームで呼び出すことができます。

重要: refreshStrategy を呼び出した後に新しい IVSLocalStageStream オブジェクトインスタンスが streamsToPublishForParticipant によって返された場合、新しいストリームオブジェクトのミュート状態がステージに適用されます。新しい IVSLocalStageStream インスタンスを作成するときは、想定どおりのミュート状態を維持するように注意してください。

リモート参加者のメディアミュート状態の監視

参加者がビデオまたはオーディオストリームのミュート状態を変更すると、変更されたストリームの配列を使用してレンダラー didChangeMutedStreams 関数が呼び出されます。IVSStageStream の isMuted プロパティを使用して、次の UI を適宜更新してください。

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

```
    }
}
```

ステージ構成を作成する

ステージの動画設定の値をカスタマイズするには、次の IVSLocalStageStreamVideoConfiguration を使用します。

```
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
```

WebRTC 統計を取得する

公開ストリームまたはサブスクライブ中のストリームの最新の WebRTC 統計情報を取得するには、IVSStageStream に requestRTCStats を使用してください。収集が完了すると、IVSStageStreamDelegate に設定できる IVSStageStream から統計を受け取ります。WebRTC の統計を継続的に収集するには、この関数を Timer で呼び出します。

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

参加者属性を取得

CreateParticipantToken エンドポイントリクエストで属性を指定すると、次のように IVSParticipantInfo プロパティにその属性が表示されます。

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

```
    }
}
```

セッションをバックグラウンドで続行

アプリがバックグラウンドに入っても、リモート音声を聞きながらステージにい続けることはできますが、自分の画像や音声を送信し続けることはできません。IVSStrategy 実装を更新して、公開を停止し、以下の.audioOnly (または .none、該当する場合) へサブスクライブする必要があります。

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

次に、stage.refreshStrategy() に電話をかけます。

サイマルキャストによるレイヤードエンコーディングの有効化/無効化

メディアストリームを公開する際、SDK は高品質と低品質のビデオストリームを伝送するため、リモートの参加者はダウンリンクの帯域幅が限られていてもストリームをサブスクライブできます。サイマルキャストによるレイヤードエンコーディングはデフォルトでオンになっています。IVSSimulcastConfiguration を使用して無効にできます。

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

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

// Other Stage implementation code
```

IVS チャネルにステージをブロードキャストする

ステージをブロードキャストするには、別の IVSBroadcastSession を作成してから、前述の SDK による通常のブロードキャスト手順に従います。IVSStageStream の device プロパティ

は、上のスニペットで示すように、IVSImageDevice または IVSAudioDevice のいずれかになります。これらを IVSBroadcastSession.mixer に接続すると、カスタマイズ可能なレイアウトでステージ全体をブロードキャストできます。

オプションで、ステージを合成して IVS 低レイテンシーチャネルにブロードキャストすることで、より多くの視聴者に届けることもできます。「IVS 低レイテンシーストリーミングユーザーガイド」の「Amazon IVS ストリームでの複数のホストの有効化」を参照してください。

iOS がカメラの解像度とフレームレートを選択する方法

Broadcast SDK によって管理されるカメラは、解像度とフレームレート (frames-per-second、または FPS) を最適化して、電力の生産と消費を最小限に抑えます。このセクションでは、ホストアプリケーションがそのユースケースに合わせて最適化できるように、解像度とフレームレートがどのように選択されるかについて説明します。

IVSCamera を使用して IVSLocalStageStream を作成するときは、フレームレート IVSLocalStageStreamVideoConfiguration.targetFramerate、および解像度 IVSLocalStageStreamVideoConfiguration.size 向けにカメラが最適化されます。IVSLocalStageStream.setConfiguration を呼び出すと、カメラが新しい値で更新されます。

カメラプレビュー

IVSCamera を IVSBroadcastSession または IVSStage にアタッチせずにそのプレビューを作成すると、デフォルト解像度の 1080p とデフォルトフレームレートの 60 fps が使用されます。

ステージのブロードキャスト

IVSBroadcastSession を使用して IVSStage をブロードキャストする場合、SDK は、両方のセッションの基準を満たす解像度とフレームレートでカメラを最適化しようとします。

例えば、ブロードキャストがフレームレート 15 FPS、解像度 1080p に設定されているときに、ステージのフレームレートが 30 FPS、解像度が 720p になっているという場合、SDK はフレームレートが 30 FPS、解像度が 1080p のカメラ設定を選択します。IVSBroadcastSession はカメラからのフレームを 1 個おきにドロップし、IVSStage は 1080p の画像を 720p に縮小します。

ホストアプリケーションが、カメラで IVSBroadcastSession と IVSStage の両方を一緒に使用する予定の場合は、それぞれの設定の targetFramerate および size プロパティを一致させることが推奨されます。一致しないプロパティは、動画のキャプチャ中にカメラがそれ自体を再設定する原因となる場合があり、これによって動画サンプルの配信がわずかに遅れます。

同一の値を設定しておくことがホストアプリケーションのユースケースに適さない場合は、最初から高品質のカメラを作成しておくことで、低品質のセッションが追加されたときにカメラがそれ自体を再設定することを防ぎます。例えば、1080p および 30 FPS でブロードキャストしてから、後ほど 720p および 30 FPS に設定されたステージに参加する場合でも、カメラはそれ自体を再設定せず、動画も中断されることなく継続します。これは、720p が 1080p 以下で、30 FPS が 30 FPS以下であるためです。

任意のフレームレート、解像度、およびアスペクト比

ほとんどのカメラハードウェアは、30 FPS で 720p、60 FPS で 1080p などの一般的なフォーマットと完全に一致します。ただし、すべてのフォーマットに完全に一致することは不可能です。Broadcast SDK は、次のルール (優先度順) に基づいてカメラ設定を選択します。

1. 解像度の幅と高さが目的の解像度以上でも、この制約内では幅と高さが可能な限り小さい。
2. フレームレートが目的のフレームレート以上でも、この制約内ではフレームレートが可能な限り低い。
3. アスペクト比は目的のアスペクト比に一致します。
4. 一致するフォーマットが複数ある場合は、視野角が最も広いフォーマットが使用されます。

ここでは、以下の 2 つの例を示します。

- ホストアプリケーションが、120 FPS、4K でブロードキャストしようとしています。選択されたカメラは、60 FPS で 4k、または 120 FPS で 1080p しかサポートしていません。フレームレートルールよりも解像度ルールが優先されるため、選択されるフォーマットは 60 FPS で 4K になります。
- 1910 x 1070 という規格外の解像度がリクエストされました。カメラは 1920 x 1080 を使用します。ご注意ください: 1921 x 1080 のような解像度を選択すると、カメラが次に利用可能な解像度 (2592 x 1944 など) にスケールアップするため、CPU およびメモリ帯域幅ペナルティが発生します。

Android の場合

Android は iOS のように解像度やフレームレートを臨機応変に調整しないため、これは Android broadcast SDK には影響しません。

既知の問題と回避策

- Bluetooth オーディオルートの変更は予測できない場合があります。セッション中に新しいデバイスを接続すると、iOS が入力ルートを自動的に変更する場合があります。また、同時に接続されている複数の Bluetooth ヘッドセットから選択することはできません。これは通常のブロードキャストとステージセッションの両方で起こります。

  回避策: Bluetooth ヘッドセットを使用する場合は、ブロードキャストまたはステージを開始する前に接続し、セッション全体を通して接続したままにします。

- 参加者が iPhone 14、iPhone 14 Plus、iPhone 14 Pro、または iPhone 14 Pro Max を使用していると、他の参加者にオーディオエコーの問題が発生する可能性があります。

  回避策: 該当するデバイスを使用している参加者は、ヘッドフォンを使用して他の参加者のエコーの問題を防ぐことができます。

- 参加者が別の参加者が使用しているトークンを使用して参加すると、最初の接続は特別なエラーなしで切断されます。

  回避策: 該当なし。

- まれにパブリッシャーが公開中でも、サブスクライバーが受け取る公開状態が `inactive` であるという問題が発生します。

  回避策:セッションを終了してから再度参加してみてください。問題が解決しない場合は、パブリッシャー用に新しいトークンを作成してください。

- 参加者が公開またはサブスクライブしている際、ネットワークが安定している場合でも、ネットワークの問題により接続が切断されたことを示すコード 1400 のエラーが表示されることがあります。

  回避策: 再パブリッシュ / 再サブスクライブを試してください。

- ステージセッション中、通常は通話時間が長くなると、まれにオーディオディストーションの問題が断続的に発生することがあります。

  回避策: オーディオディストーションの問題が起きる参加者は、セッションを終了してから再度参加するか、音声を非公開にしてから再公開して問題を解決できます。

エラー処理

致命的なエラーと致命的でないエラー

エラーオブジェクトには、「is fatal」ブール値が含まれています。これはブール値を含む IVSBroadcastErrorIsFatalKey 内のディクショナリエントリです。

一般的に、致命的なエラーは Stages サーバーへの接続に関連しています (接続を確立できないか、接続が失われて回復できないかのいずれか)。アプリケーションはステージを再作成し、(場合によっては新しいトークンを使って、あるいはデバイスの接続が回復したときに) 再参加する必要があります。

致命的でないエラーは通常、公開/サブスクライブ状態に関連しており、公開/サブスクライブ操作を再試行する SDK によって処理されます。

次のプロパティを確認できます。

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

参加エラー

不正な形式のトークン

これは、ステージトークンの形式が不正な場合に発生します。

SDK は、エラーコード = 1000 および IVS BroadcastErrorIsFatalKey = Yes で Swift 例外をスローします。

アクション: 有効なトークンを作成し、参加を再試行してください。

期限切れのトークン

これは、ステージトークンの有効期限が切れた場合に発生します。

SDK は、エラーコード = 1001、IVS BroadcastErrorIsFatalKey = Yes で Swift 例外をスローします。

アクション: 新しいトークンを作成し、参加を再試行してください。

無効または取り消されたトークン

これは、ステージトークンの形式が不正でないにもかかわらず、Stages サーバーによって拒否された場合に発生します。これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK は、エラーコード = 1026、IVS BroadcastErrorIsFatalKey = Yes `stage(didChange connectionState, withError error)`で を呼び出します。

アクション: 有効なトークンを作成し、参加を再試行してください。

初期参加におけるネットワークエラー

これは、SDK が Stages サーバーにアクセスしたにもかかわらず、接続を確立できなかった場合に発生します。これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK は、エラーコード = 1300、IVS BroadcastErrorIsFatalKey = Yes `stage(didChange connectionState, withError error)`で を呼び出します。

アクション: デバイスの接続が回復するのを待ってから、参加を再試行してください。

既に参加している場合のネットワークエラー

デバイスのネットワーク接続が切断されると、SDK と Stages サーバーとの接続が失われる可能性があります。これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK は、エラーコード = 1300、IVS BroadcastErrorIsFatalKey 値 = Yes `stage(didChange connectionState, withError error)`で を呼び出します。

アクション: デバイスの接続が回復するのを待ってから、参加を再試行してください。

公開/サブスクライブエラー

初期

次のようなエラーがあります。

- MultihostSessionOfferCreationFailPublish (1020)

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

これらは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK は限られた回数だけ操作を再試行します。再試行中の公開/サブスクライブ状態は ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE です。再試行が成功すると、状態は PUBLISHED/SUBSCRIBED に変更されます。

SDK は、関連するエラーコードと IVS BroadcastErrorIsFatalKey = NO IVSErrorSourceDelegate:didEmitErrorを使用して を呼び出します。

アクション: SDK は自動的に再試行するため、アクションは不要です。オプションで、アプリケーションはストラテジーを更新して再試行回数を増やすことができます。

確立済みにもかかわらず失敗する

公開またはサブスクライブは、確立された後に失敗することがあります (ネットワークエラーが原因である可能性が高い)。「ネットワークエラーによりピア接続が失われた」ことを示すエラーコードは 1400 です。

これは、アプリケーションに搭載されているステージレンダラーを介して非同期的に報告されます。

SDK は、公開/サブスクライブ操作を再試行します。再試行中の公開/サブスクライブ状態は ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE です。再試行が成功すると、状態は PUBLISHED/SUBSCRIBED に変更されます。

SDK は、エラーコード = 1400、IVS BroadcastErrorIsFatalKey = NO didEmitErrorで を呼び出します。

アクション: SDK は自動的に再試行するため、アクションは不要です。オプションで、アプリケーションはストラテジーを更新して再試行回数を増やすことができます。接続が完全に失われた場合、Stages への接続も失敗する可能性があります。

IVS Broadcast SDK: カスタムイメージソース (リアルタイムストリーミング)

カスタム画像入力ソースを使用することで、プリセットされたカメラに限定されるのではなく、アプリケーションが独自の画像入力を Broadcast SDK に提供できます。カスタム画像ソースは、半透明の透かしや静的な「be right back」(すぐに戻ります) シーンと同じくらいシンプルにすることも、アプリケーションがカメラにビューティーフィルターを追加するなど、追加のカスタム処理を実行できるようにすることもできます。

カメラのカスタムコントロール (カメラアクセスを必要とする加工フィルターライブラリを使用するなど) のためにカスタム画像入力ソースを使用する場合、ブロードキャスト SDK はカメラの管理に責任を負わなくなります。代わりに、アプリケーションは、カメラのライフサイクルの正しい処理に責任を負うようになります。アプリケーションがカメラをどのように管理すべきかについては、プラットフォームの公式ドキュメントを参照してください。

Android

`DeviceDiscovery` セッションを作成したら、画像入力ソースを作成します。

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

このメソッドは、標準の Android Surface に基づく画像ソースである `CustomImageSource` を返します。サブクラス `SurfaceSource` のサイズを変更したり、回転したりできます。`ImagePreviewView` を作成して、その内容のプレビューを表示することもできます。

基盤の `Surface` を取得するには:

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

この `Surface` は、Camera2、OpenGL ES、他のライブラリなどの画像プロデューサーの出力バッファとして使用できます。最も簡単なユースケースは、静的なビットマップまたは色を Surface のキャンバスに直接描画することです。ただし、多くのライブラリ (ビューティーフィルターライブラリなど) は、アプリケーションがレンダリングのために外部の `Surface` を指定できるようにするメソッドを提供します。このようなメソッドを使用して、この `Surface` をフィルターライブラリに渡すことができます。これにより、ライブラリは、ブロードキャストセッションのために処理されたフレームを出力してストリーミングできます。

この CustomImageSource は LocalStageStream でラップすることができ、StageStrategy によって返されて Stage に公開されます。。

iOS

DeviceDiscovery セッションを作成したら、画像入力ソースを作成します。

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

このメソッドは、アプリケーションが手動で CMSampleBuffers を送信できるようにする画像ソースである IVSCustomImageSource を返します。サポートされているピクセル形式については、「iOS ブロードキャスト SDK リファレンス」を参照してください。最新バージョンへのリンクは、最新のブロードキャスト SDK リリースの「Amazon IVS リリースノート」にあります。

カスタムソースに送信されたサンプルは、次のステージでストリーミングされます。

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

動画のストリーミングには、このメソッドをコールバックで使用します。例えば、カメラを使用している場合、AVCaptureSession から新しいサンプルバッファを受信するたびに、アプリケーションはサンプルバッファをカスタム画像ソースに転送できます。必要に応じて、カスタム画像ソースにサンプルを送信する前に、アプリケーションでさらなる処理 (ビューティーフィルターなど) を適用できます。

IVSCustomImageSource は IVSLocalStageStream でラップすることができ、IVSStageStrategy によって返されて Stage に公開されます。。

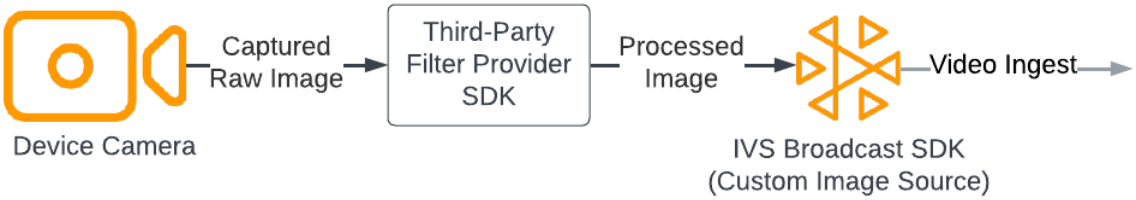
IVS Broadcast SDK: サードパーティーのカメラフィルター (リアルタイムストリーミング)

このガイドは、読者が既にカスタム画像ソースに慣れていて IVS Broadcast SDK (リアルタイムストリーミング) をアプリケーションに統合していることを前提としています。

カメラフィルターを使うと、ライブストリームのクリエイターは顔や背景の見た目を補強したり変更したりできます。これにより、視聴者の注目度を高め、視聴者を引き付け、ライブストリーミングに好感を持たせることができる可能性があります。

サードパーティーのカメラフィルターを統合する

フィルター SDK の出力をカスタム画像入力ソースにフィードすることで、サードパーティーのカメラフィルター SDK を IVS ブロードキャスト SDK と統合できます。カスタム画像入力ソースを使用することで、アプリケーションが独自の画像入力を Broadcast SDK に提供できます。サードパーティーのフィルタープロバイダーの SDK がカメラのライフサイクルを管理して、カメラからの画像の処理、フィルター効果の適用、カスタム画像ソースに渡せる形式で出力する場合があります。



カメラフレームにフィルター効果を適用してカスタム画像入力ソースに渡せる形式に変換する組み込みメソッドについては、サードパーティーのフィルタープロバイダーのドキュメントを参照してください。この処理は、使用している IVS ブロードキャスト SDK のバージョンによって異なります。

- ウェブ – フィルタープロバイダーは、出力をキャンバス要素にレンダリングできる必要があります。そうすると、captureStream メソッドを使用して、キャンバスのコンテンツの MediaStream を返すことができます。そして、MediaStream を LocalStageStream のインスタンスに変換して、ステージに公開できます。
- Android – フィルタープロバイダーの SDK は、IVS Broadcast SDK が提供する Android `Surface` にフレームをレンダリングすることも、フレームをビットマップに変換することもできます。ビットマップを使用する場合は、ロックを解除してキャンバスに書き込むことで、カスタム画像ソースが提供する基盤 `Surface` にレンダリングできます。
- iOS – サードパーティーのフィルタープロバイダーの SDK は、フィルター効果を適用したカメラフレームを `CMSampleBuffer` として提供する必要があります。カメラ画像の処理後に `CMSampleBuffer` を最終出力として使用する方法については、サードパーティーのフィルターベンダーの SDK のドキュメントを参照してください。

BytePlus

Android

BytePlus Effects SDK のインストールと設定

BytePlus Effects SDK のインストール、初期化、設定方法の詳細については、BytePlus「Android アクセスガイド」を参照してください。

カスタム画像ソースを設定する

SDK を初期化した後、フィルター効果を適用した処理済みのカメラフレームをカスタム画像入力ソースにフィードします。そのためには、DeviceDiscovery オブジェクトのインスタンスを作成し、カスタム画像ソースを作成します。カメラのカスタムコントロールにカスタム画像入力ソースを使用する場合、Broadcast SDK はカメラの管理をしなくなることに注意してください。代わりに、アプリケーションはカメラのライフサイクルを正しく処理する責任があります。

Java

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

出力をビットマップに変換し、カスタム画像入力ソースにフィードする

BytePlus Effects SDK からフィルター効果が適用されたカメラフレームを IVS Broadcast SDK に直接転送できるようにするには、BytePlus Effects SDK のテクスチャ出力をビットマップに変換します。画像が処理されると、SDK によって onDrawFrame() メソッドが呼び出されます。onDrawFrame() メソッドは Android の GLSurfaceView.Renderer インターフェイスのパブリックメソッドです。BytePlus が提供する Android サンプルアプリケーションでは、このメソッドはカメラフレームごとに呼び出され、テクスチャを出力します。同時に、このテクスチャをビットマップに変換してカスタム画像入力ソースにフィードするロジックを onDrawFrame() メソッドに追加できます。次のサンプルコードに示すように、BytePlus SDK が提供する transferTextureToBitmap メソッドを使用してこの変換を行います。このメソッドは、次のサンプルコードに示すように、BytePlus Effects SDK から com.bytedance.labcv.core.util.ImageUtil ライブラリで提供されます。次に、結果のビットマップを Surface のキャンバスに書き込むことで、CustomImageSource の基盤 Android Surface にレンダリングできます。onDrawFrame() を何度も連続して呼び出すと、ビットマップのシーケンスが生成され、組み合わせると動画のストリームが作成されます。

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

DeepAR SDK を Android IVS Broadcast SDK と統合する方法の詳細については、DeepAR の Android 統合ガイドを参照してください。

iOS

DeepAR SDK を iOS IVS Broadcast SDK と統合する方法の詳細については、DeepAR の iOS 統合ガイドを参照してください。

Snap

Web

このセクションは、読者が既に Web Broadcast SDK を使用した動画の公開とサブスクリプションに慣れていることを前提としています。

Snap の Camera Kit SDK を IVS リアルタイムストリーミング Web Broadcast SDK と統合するには、以下が必要です。

1. Camera Kit SDK と Webpack をインストールします。(この例では Webpack をバンドラーとして使用していますが、任意のバンドラーを使用できます)
2. `index.html` を作成します。

3. セットアップ要素を追加します。
4. 参加者を表示して設定します。
5. 接続されているカメラとマイクを表示します。
6. Camera Kit セッションを作成します。
7. Lens を取得して適用します。
8. Camera Kit セッションの出力をキャンバスにレンダリングします。
9. Camera Kit にレンダリング用のメディアソースを供給し、`LocalStageStream` を公開します。
10. Webpack 設定ファイルを作成します。

各ステップの詳細を以下に示します。

Camera Kit SDK と Webpack をインストールする

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

index.html を作成する

次に、HTML 共通スクリプトを作成し、Web Broadcast SDK をスクリプトタグとしてインポートします。次のコードでは、<SDK version> を、使用している Broadcast SDK のバージョンに必ず置き換えてください。

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>
```

セットアップ要素を追加する

カメラとマイクを選択し、参加者トークンを指定するための HTML を次のように作成します。

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>
```

その下に HTML を追加して、ローカルおよびリモートの参加者からのカメラフィードを表示します。

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>
```

カメラをセットアップするためのヘルパーメソッドなどの追加のロジックやバンドルされている JavaScript ファイルをロードします。(このセクションの後半では、これらの JavaScript ファイルを作成して 1 つのファイルにバンドルします。これにより、Camera Kit をモジュールとしてインポートできます。バンドルされている JavaScript ファイルには、Camera Kit の設定、Lens の適用、およびステージにレンズを適用したカメラフィードの公開を行うためのロジックが含まれます)

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>
```

参加者を表示および設定する

次に helpers.js を作成します。これには、参加者の表示と設定に使用するヘルパーメソッドが含まれます。

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;
}
```

接続されたカメラとマイクを表示する

次に media-devices.js を作成します。これには、デバイスに接続されているカメラとマイクを表示するためのヘルパーメソッドが含まれます。

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,
    },
  });
}
```

Camera Kit セッションを作成する

`stages.js` を作成します。これには、カメラフィードに Lens を適用し、そのフィードをステージに公開するためのロジックが含まれます。このファイルの最初の部分では、Broadcast SDK と Camera Kit Web SDK をインポートし、各 SDK で使用する変数を初期化します。Camera Kit Web SDK をブートストラップした後に `createSession` を呼び出すことで、Camera Kit セッションを作成します。キャンバス要素オブジェクトがセッションに渡されることに注意してください。これにより、Camera Kit はそのキャンバスにレンダリングするよう指示されます。

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 });
```

Lens を取得して適用する

Lens を取得するには、Camera Kit デベロッパーポータルにある Lens グループ ID を挿入します。この例では、返される Lens 配列の最初の Lens を適用することで簡単にしています。

JavaScript

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

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

Camera Kit セッションからの出力をキャンバスにレンダリングする

captureStream メソッドを使用して、キャンバスのコンテンツの MediaStream を返します。キャンバスには、Lens が適用されたカメラフィードのビデオストリームが含まれます。また、カメラとマイクをミュートするボタン用のイベントリスナーと、ステージに参加したりステージから退出したりするためのイベントリスナーも追加します。ステージに参加するためのイベントリスナーでは、Camera Kit セッションとキャンバスからの MediaStream を渡して、ステージに公開できるようにします。

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();
  });
};
```

Camera Kit にレンダリング用のメディアソースを供給し、LocalStageStream を公開します。

Lens を適用したビデオストリームを公開するには、以前にキャンバスからキャプチャした MediaStream を渡す setCameraKitSource という名前の関数を作成します。ローカルカメラフィードをまだ組み込んでいないので、キャンバスからの MediaStream は今のところ何もしていません。getCamera ヘルパーメソッドを呼び出して localCamera に割り当てることで、ローカルカメラフィードを組み込むことができます。その後、ローカルカメラフィード (localCamera 経由) とセッションオブジェクトを setCameraKitSource に渡すことができます。setCameraKitSource 関数は createMediaStreamSource の呼び出しによってローカルカメラフィードを CameraKit のメディアソースに変換します。次に、CameraKit のメディアソースは前面カメラをミラーリングするように変換されます。次に、Lens 効果がメディアソースに適用され、session.play() の呼び出しによって出力キャンバスにレンダリングされます。

キャンバスからキャプチャした MediaStream に Lens が適用されたので、ステージへの公開に進むことができます。そのためには、MediaStream のビデオトラックを使用して LocalStageStream を作成します。その後、LocalStageStream のインスタンスを StageStrategy に渡して公開できます。

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;
    },
  };
```

以下の残りのコードは、ステージを作成および管理するためのものです。

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();
```

Webpack 設定ファイルを作成する

webpack.config.js を作成して次のコードを追加します。これにより上記のロジックがバンドルされ、import ステートメントを使用して Camera Kit を使用できるようになります。

JavaScript

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

最後に、npm run build を実行して、Webpack 設定ファイルで定義されている JavaScript をバンドルします。これで、ウェブサーバーから HTML と JavaScript を配信できます。例えば、次のように Python の HTTP サーバーを使用して localhost:8000 を開くと、結果を確認できます。

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

Android

Snap の Camera Kit SDK を IVS Android Broadcast SDK と統合するには、Camera Kit SDK をインストールし、Camera Kit セッションを初期化し、Lens を適用して、Camera Kit セッションの出力をカスタム画像入力ソースに送る必要があります。

Camera Kit SDK をインストールするには、モジュールの build.gradle ファイルに以下を追加します。$cameraKitVersion を Camera Kit SDK の最新バージョンに置き換えます。

Java

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

cameraKitSession を初期化および取得します。Camera Kit には Android の CameraX API 用の便利なラッパーも用意されているため、Camera Kit で CameraX を使用する際に複雑なロジックを記述する必要はありません。CameraXImageProcessorSource オブジェクトを ImageProcessor のソースとして使用して、カメラプレビューストリーミングフレームを開始できます。

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();
   }
```

Lens を取得して適用する

Camera Kit デベロッパーポータルで、Lens とそのカルーセル内での順序を設定できます。

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

ブロードキャストするには、処理済みのフレームをカスタム画像ソースの基盤 Surface に送信します。DeviceDiscovery オブジェクトを使用して CustomImageSource を作成し、SurfaceSource を返します。その後、CameraKit セッションからの出力を SurfaceSource が提供する基盤 Surface にレンダリングできます。

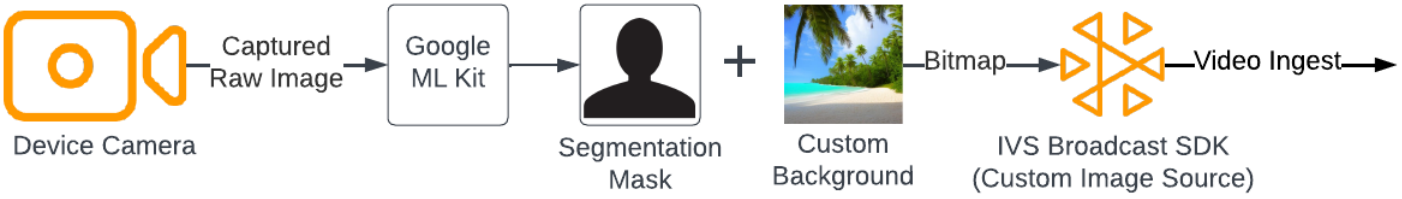
Java

```
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
```

背景の置換

背景の置換は、ライブストリームのクリエイターが背景を変更できるようにするカメラフィルターの一種です。次の図に示すように、背景の置換には以下が必要です。

1. ライブカメラフィードからカメラ画像を取得します。
2. Google ML Kit を使用して前景コンポーネントと背景コンポーネントに分割します。
3. 生成された分割マスクをカスタムの背景画像と組み合わせます。
4. それをカスタム画像ソースに渡してブロードキャストします。



Web

このセクションは、読者が既に Web Broadcast SDK を使用した動画の公開とサブスクリプションに慣れていることを前提としています。

ライブストリームの背景をカスタム画像に置き換えるには、MediaPipe Image Segmenter で selfie segmentation model を使用します。これは動画フレーム内のどのピクセルが前景か背景かを識別する機械学習モデルです。その後、ビデオフィードの前景ピクセルを新しい背景を表すカスタム画像にコピーすることで、モデルの結果を使用してライブストリームの背景を置き換えることができます。

背景の置換を IVS リアルタイムストリーミング Web Broadcast SDK と統合するには、以下が必要です。

1. MediaPipe と Webpack をインストールします。(この例では Webpack をバンドラーとして使用していますが、任意のバンドラーを使用できます)
2. 作成 `index.html`。
3. メディア要素を追加します。
4. スクリプトタグを追加します。
5. 作成 `app.js`。
6. カスタム背景画像を読み込みます。
7. `ImageSegmenter` のインスタンスを作成します。
8. ビデオフィードをキャンバスにレンダリングします。
9. 背景置換ロジックを作成します。
10. Webpack 設定ファイルを作成します。
11. JavaScript ファイルをバンドルします。

MediaPipe と Webpack をインストールする

まず、`@mediapipe/tasks-vision` と `webpack` npm パッケージをインストールします。以下の例では、Webpack を JavaScript バンドラーとして使用しています。必要に応じて別のバンドラーを使用できます。

JavaScript

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

また、次のように、ビルドスクリプトとして webpack を指定するように package.json も必ず更新します。

JavaScript

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

index.html を作成する

次に、HTML 共通スクリプトを作成し、Web Broadcast SDK をスクリプトタグとしてインポートします。次のコードでは、<SDK version> を、使用している Broadcast SDK のバージョンに必ず置き換えてください。

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>
```

メディア要素を追加する

次に、ボディタグ内にビデオ要素と 2 つのキャンバス要素を追加します。ビデオ要素にはライブカメラフィードが含まれ、MediaPipe Image Segmenter への入力として使用されます。最初のキャンバス要素は、ブロードキャストされるフィードのプレビューをレンダリングするために使用されます。2 番目のキャンバス要素は、背景として使用されるカスタム画像のレンダリングに使用されます。カスタム画像を含む 2 番目のキャンバスは、そこから最終的なキャンバスにピクセルをプログラムでコピーするためのソースとしてのみ使用されるため、表示されなくなります。

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>
```

スクリプトタグを追加する

スクリプトタグを追加して、背景の置換を行うコードを含むバンドルされた JavaScript ファイルをロードし、ステージに公開します。

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

app.js の作成

次に、JavaScript ファイルを作成して、HTML ページに作成されたキャンバス要素とビデオ要素の要素オブジェクトを取得します。`ImageSegmenter` モジュールと `FilesetResolver` モジュールをインポートします。`ImageSegmenter` モジュールは分割タスクの実行に使用されます。

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";
```

次に、ユーザーのカメラから MediaStream を取得する `init()` という関数を作成し、カメラフレームの読み込みが完了するたびにコールバック関数を呼び出します。ステージに参加したりステージから退出したりするボタンのイベントリスナーを追加します。

ステージに参加するときは、`segmentationStream` という名前の変数を渡すことに注意してください。これはキャンバス要素からキャプチャされたビデオストリームで、背景を表すカスタム画像に前景画像が重なっています。その後、このカスタムストリームを使用して `LocalStageStream` のインスタンスを作成し、ステージに公開できます。

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();
  });
};
```

カスタム背景画像を読み込む

init 関数の下部に、initBackgroundCanvas という名前の関数を呼び出すコードを追加します。これにより、ローカルファイルからカスタム画像が読み込まれ、キャンバスにレンダリングされます。この関数は次のステップで定義します。ユーザーのカメラから取得した MediaStream をビデオオブジェクトに割り当てます。その後、このビデオオブジェクトは Image Segmenter に渡されます。また、renderVideoToCanvas という名前の関数をビデオフレームの読み込みが完了するたびに呼び出されるコールバック関数として設定します。この関数は後のステップで定義します。

JavaScript

```
initBackgroundCanvas();

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

ローカルファイルから画像を読み込む initBackgroundCanvas 関数を実装しましょう。この例では、カスタム背景としてビーチの画像を使用します。カスタム画像を含むキャンバスは、カメラフィードを含むキャンバス要素の前景ピクセルとマージされるため、表示されなくなります。

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);
  };
};
```

ImageSegmenter のインスタンスを作成する

次に、ImageSegmenter のインスタンスを作成します。これにより、画像が分割され、その結果がマスクとして返されます。ImageSegmenter のインスタンスを作成するときは、selfie segmentation model を使用します。

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,
  });
};
```

ビデオフィードをキャンバスにレンダリングする

次に、ビデオフィードを他のキャンバス要素にレンダリングする関数を作成します。ビデオフィードをキャンバスにレンダリングする必要があります。そうすると、Canvas 2D API を使用してそこから前景ピクセルを抽出できるようになります。また、その際、segmentForVideo メソッドを使用してビデオフレームを ImageSegmenter のインスタンスに渡し、ビデオフレーム内の前景と背景を分割します。segmentForVideo メソッドが戻ると、背景の置換を行うためのカスタムコールバック関数である replaceBackground が呼び出されます。

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);
};
```

背景の置換ロジックを作成する

カスタム背景画像をカメラフィードの前景と結合して背景を置き換える replaceBackground 関数を作成します。この関数はまず、先に作成した 2 つのキャンバス要素から、カスタム背景画像とビデオフィードの基盤ピクセルデータを取得します。次に、ImageSegmenter から提供されたマスクを繰り返し適用します。これにより、どのピクセルが前景にあるかがわかります。マスクを繰り返し適用しながら、ユーザーのカメラフィードを含むピクセルを、対応する背景ピクセルデータに選択的にコピーします。これが完了すると、前景がコピーされた最終的なピクセルデータを背景に変換し、キャンバスに描画します。

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);
}
```

参考までに、上記のすべてのロジックを含む完全な app.js ファイルを次に示します。

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();
```

Webpack 設定ファイルを作成する

次の設定を Webpack 設定ファイルに追加して app.js をバンドルすると、インポート呼び出しが動作するようになります。

JavaScript

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

JavaScript ファイルをバンドルする

```
npm run build
```

index.html を含むディレクトリから単純な HTTP サーバーを起動し、localhost:8000 を開いて結果を確認します。

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

Android

ライブストリームの背景を置換するには、Google ML Kit の selfie segmentation API を使用できます。selfie segmentation API はカメラ画像を入力として受け入れ、画像の各ピクセルの信頼度スコアを示すマスクを返します。これにより、画像のピクセルが前景にあったか背景にあったかがわかります。信頼度スコアに基づいて、背景画像または前景画像から対応するピクセルの色を取得できます。この処理は、マスク内のすべての信頼度スコアが検証されるまで続きます。その結果、背景画像のピクセルと前景のピクセルを組み合わせた新しいピクセルの色の配列が生成されます。

背景の置換を IVS リアルタイムストリーミング Android Broadcast SDK と統合するには、以下が必要です。

1. CameraX ライブラリと Google ML Kit をインストールします。
2. 共通スクリプト変数を初期化します。
3. カスタム画像ソースを作成します。
4. カメラフレームを管理します。
5. カメラフレームを Google ML Kit に渡します。
6. カメラフレームの前景をカスタム背景に重ねます。
7. 新しい画像をカスタム画像ソースにフィードします。

CameraX ライブラリと Google ML Kit をインストールする

ライブカメラフィードから画像を抽出するには、Android の CameraX ライブラリを使用します。CameraX ライブラリと Google ML Kit をインストールするには、モジュールの build.gradle ファイルに以下を追加します。${camerax_version} と ${google_ml_kit_version} をそれぞれ CameraX ライブラリと Google ML Kit ライブラリの最新バージョンに置き換えます。

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}"
```

以下のライブラリをインポートします。

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
```

共通スクリプト変数を初期化する

ImageAnalysis のインスタンスと ExecutorService のインスタンスを初期化します。

Java

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

Segmenter インスタンスを STREAM_MODE に初期化します。

Java

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

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

カスタム画像ソースを作成する

アクティビティの onCreate メソッドで、DeviceDiscovery オブジェクトのインスタンスを作成し、カスタム画像ソースを作成します。カスタム画像ソースから提供された Surface が、カスタム背景画像に前景が重なった最終イメージを受け取ります。次に、カスタム画像ソースを使用して ImageLocalStageStream のインスタンスを作成します。その後、ImageLocalStageStream のインスタンス (この例では名前は filterStream) をステージに公開できます。ステージの設定方法

については、「IVS Broadcast SDK: Android ガイド」を参照してください。最後に、カメラの管理に使用するスレッドも作成します。

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()
```

カメラフレームを管理する

次に、カメラを初期化する関数を作成します。この関数は CameraX ライブラリを使用して、ライブカメラフィードから画像を抽出します。まず、cameraProviderFuture という ProcessCameraProvider のインスタンスを作成します。このオブジェクトは、カメラプロバイダーを取得した将来の結果を表します。次に、プロジェクトから画像をビットマップとして読み込みます。この例では、ビーチの画像を背景として使用していますが、どのような画像でもかまいません。

次に、cameraProviderFuture にリスナーを追加します。このリスナーには、カメラが使用可能になったとき、またはカメラプロバイダーの取得中にエラーが発生した場合に通知されます。

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))
    }
```

リスナー内で、ライブカメラフィードから個々のフレームにアクセスするように ImageAnalysis.Builder を作成します。バックプレッシャーストラテジーを STRATEGY_KEEP_ONLY_LATEST に設定します。これにより、一度に 1 つのカメラフレームだけが処理に送られることが保証されます。個々のカメラフレームをビットマップに変換すると、そのピクセルを抽出して後でカスタム背景画像と組み合わせることができます。

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())
```

カメラフレームを Google ML Kit に渡す

次に、InputImage を作成して Segmenter のインスタンスに渡して処理します。InputImage は、ImageAnalysis のインスタンスから提供された ImageProxy から作成できます。Segmenter に InputImage が提供されると、ピクセルが前景または背景にある可能性を示す信頼度スコア付きのマスクが返されます。このマスクには幅と高さのプロパティもあります。これを使用して、前に読み込んだカスタム背景画像の背景ピクセルを含む新しい配列を作成します。

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)
```

カメラフレームの前景をカスタム背景に重ねる

信頼度スコアを含むマスク、ビットマップとしてのカメラフレーム、カスタム背景画像のカラーピクセルがあれば、前景をカスタム背景に重ねるのに必要なものがすべて揃っています。次に、overlayForeground 関数が以下のパラメータで呼び出されます。

Java

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

この関数はマスクを繰り返し適用し、信頼度の値をチェックして、対応するピクセルの色を背景画像から取得するのか、カメラフレームから取得するのかを決定します。マスク内のピクセルが背景にある可能性が高いことを信頼度の値が示している場合、対応するピクセルの色を背景画像から取得します。それ以外の場合は、カメラフレームから対応するピクセルの色を取得して前景を構築します。関数がマスクの反復適用を終了すると、新しいカラーピクセルの配列を使用して新しいビットマップが作成され、返されます。この新しいビットマップには、カスタム背景に重なった前景が含まれています。

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
        )
    }
```

新しい画像をカスタム画像ソースにフィードする

その後、カスタム画像ソースから提供された Surface に新しいビットマップを書き込むことができます。これでステージにブロードキャストされます。

Java

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

カメラフレームを取得して Segmenter に渡し、背景に重ねる関数をすべて次に示します。

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))
    }
```

IVS Broadcast SDK: モバイルオーディオモード (リアルタイムストリーミング)

オーディオ品質はリアルチームのメディアエクスペリエンスにとって重要な部分であり、あらゆるユースケースに最適な万能のオーディオ構成はありません。IVS リアルタイムストリームを聴くときにユーザーが最高の体験を得られるように、モバイル SDK には複数のプリセットオーディオ構成が用意されているほか、必要に応じてさらに強力なカスタマイズも可能です。

序章

IVS モバイル Broadcast SDK には `StageAudioManager` クラスが用意されています。このクラスは、2 つのプラットフォームの基盤オーディオモードを一元的に制御できるように設計されています。Android では、オーディオモード、オーディオソース、コンテンツタイプ、使用方法、通信デバイスなど、AudioManager を制御します。iOS では、アプリケーションの AVAudioSession を制御し、voiceProcessing を有効にするかどうかも制御します。

重要: IVS リアルタイム Broadcast SDK がアクティブな間は、`AVAudioSession` や `AudioManager` を直接操作しないでください。これを行うと、オーディオが失われたり、間違ったデバイスから録音されたり、間違ったデバイスで再生されたりする可能性があります。

最初の `DeviceDiscovery` オブジェクトまたは `Stage` オブジェクトを作成する前に、`StageAudioManager` クラスを設定する必要があります。

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
```

DeviceDiscovery インスタンスや Stage インスタンスの初期化前に StageAudioManager に何も設定されていない場合、VideoChat プリセットが自動的に適用されます。

オーディオモードプリセット

リアルタイム Broadcast SDK には 3 つのプリセットがあり、以下で説明するように、それぞれ一般的なユースケースに合わせて調整されています。各プリセットについて、プリセットを互いに区別する 5 つの主要カテゴリについて説明します。

ビデオチャット

これはデフォルトのプリセットで、ローカルデバイスが他の参加者とリアルタイムで会話する場合に適しています。

カテゴリ	Android	iOS
エコーキャンセレーション	有効	有効
Volume Rocker	通話ボリューム	通話ボリューム
マイク選択	OS によって制限されます。USB マイクを使用できない場合があります。	OS によって制限されます。USB マイクと Bluetooth マイクを使用できない場合があります。 AirPods のように、入力と出力の両方を同時に処理する Bluetooth ヘッドセットは動作するはずです。

カテゴリ	Android	iOS
オーディオ出力	どの出力デバイスでも動作するはずです。	OS によって制限されます。有線ヘッドセットを使用できない場合があります。
オーディオ品質	中/低 メディア再生とは違い、電話のように聞こえます。	中/低 メディア再生とは違い、電話のように聞こえます。

サブスクライブのみ

このプリセットは、公開している他の参加者をサブスクライブする予定があっても自分では公開しない場合向けに設計されています。オーディオの品質に重点を置き、利用可能なすべての出力デバイスをサポートしています。

カテゴリ	Android	iOS
エコーキャンセレーション	無効	無効
Volume Rocker	メディアボリューム	メディアボリューム
マイク選択	非該当。このプリセットは公開用ではありません。	非該当。このプリセットは公開用ではありません。
オーディオ出力	どの出力デバイスでも動作するはずです。	どの出力デバイスでも動作するはずです。
オーディオ品質	高 音楽を含め、どんな種類のメディアでもクリアに聞こえるはずです。	高 音楽を含め、どんな種類のメディアでもクリアに聞こえるはずです。

Studio

このプリセットは、公開機能を維持したまま、質の高いサブスクライブができるように設計されています。エコーキャンセレーション機能付きの録音再生ハードウェアが必要です。この場合のユース

ケースは、USB マイクと有線ヘッドセットの使用です。エコーの原因とならないようにデバイスの物理的な分離を確保しながら、SDK は最高品質のオーディオを維持します。

カテゴリ	Android	iOS
エコーキャンセレーション	無効	無効
Volume Rocker	ほとんどの場合、メディアボリューム。Bluetooth マイクが接続されている場合は、通話ボリューム。	メディアボリューム
マイク選択	どのマイクでも動作するはずです。	どのマイクでも動作するはずです。
オーディオ出力	どの出力デバイスでも動作するはずです。	どの出力デバイスでも動作するはずです。
オーディオ品質	高 双方が音楽を送信でき、反対側でもクリアに聞こえるはずです。 Bluetooth ヘッドセットが接続されている場合、Bluetooth SCO モードが有効になっているため、音質が低下します。	高 双方が音楽を送信でき、反対側でもクリアに聞こえるはずです。 Bluetooth ヘッドセットが接続されている場合、ヘッドセットによっては Bluetooth SCO モードが有効になっているため、音質が低下する場合があります。

高度なユースケース

プリセット以外にも、iOS と Android のリアルタイムストリーミング Broadcast SDK では、基盤となるプラットフォームのオーディオモードを次のように設定できます。

- Android では、AudioSource、Usage、ContentType を設定します。
- iOS では、AVAudioSession.Category、AVAudioSession.CategoryOptions、AVAudioSession.Mode、および公開中に音声処理を有効にするかどうかを切り替える機能を使用します。

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
```

Android で Bluetooth を使用して公開する

以下の条件が満たされると、SDK は Android の `VIDEO_CHAT` プリセットに自動的に戻ります。

- 割り当てられた設定には、`VOICE_COMMUNICATION` 使用状況の値は使用されていない。
- Bluetooth マイクがデバイスに接続されている。
- ローカル参加者がステージに公開している。

これは Bluetooth ヘッドセットを使用してオーディオを録音する方法に関する Android オペレーティングシステムの制限です。

他の SDK との統合

iOS と Android はどちらもアプリケーションごとにアクティブなオーディオモードを 1 つしかサポートしていないため、オーディオモードの制御を必要とする複数の SDK をアプリケーションで使用していると、競合が発生することがよくあります。このような競合が発生した場合は、以下で説明する一般的な解決方法をいくつか試してみてください。

オーディオモードの値に合わせる

IVS SDK の高度なオーディオ設定オプションまたは他の SDK の機能を使用して、2 つの SDK を基本となる値に合わせます。

Agora

iOS

iOS では、Agora SDK に `AVAudioSession` をアクティブのままにしておくように指示すると、IVS リアルタイムストリーミング Broadcast SDK が使用している間は非アクティブ化されなくなります。

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

Android

IVS リアルタイムストリーミング Broadcast SDK で公開中に `RtcEngine` で `setEnableSpeakerphone` を呼び出すことや `enableLocalAudio(false)` を呼び出すことは避けてください。IVS SDK で公開していないときに再度 `enableLocalAudio(true)` を呼び出す機会があります。

IVS リアルタイムストリーミングで Amazon EventBridge を使用する

Amazon EventBridge を使用して、Amazon Interactive Video Service (IVS) ストリームをモニタリングできます。

Amazon IVS は、ストリームのステータスに関する変更イベントを Amazon EventBridge に送信します。配信されたすべてのイベントが有効です。ただし、イベントはベストエフォートベースで送信されます。つまり、以下を保証するものではありません。

- イベントが配信される – 指定されたイベントの実行 (参加者による公開など) は可能ですが、Amazon IVS は、対応するイベントを EventBridge に送信しないことがあります。Amazon IVS は、配信を中止する前に、数時間にわたりイベントの配信を試みます。
- 配信されたイベントが指定された時間内に到着する — 数時間前のイベントを受け取ることもあります。
- イベントが順序通りに配信される – イベントは、特に短い時間内に送信された場合、順不同になることがあります。例えば、Participant Published の前に Participant Unpublished が送信される場合があります。

イベントが欠落したり、遅延したり、順序が違ったりすることはまれですが、通知イベントの順序や存在に依存するビジネスクリティカルなプログラムを作成するときは、こうした可能性に対処しておく必要があります。

EventBridge ルールは、以下のイベントに対して作成できます。

イベントタイプ	イベント	配信するタイミング
IVS コンポジションの状態変化	送信先障害	送信先への出力に失敗しました。例えば、ストリームキーがないか、別のブロードキャストが進行中の場合には、あるチャンネルへのブロードキャストが失敗します。
IVS コンポジションの状態変化	送信先の開始	送信先への出力が正常に開始されました。

イベントタイプ	イベント	配信するタイミング
IVS コンポジションの状態変化	送信先の終了	送信先への出力が完了しました。
IVS コンポジションの状態変化	送信先の再接続	送信先への出力が中断されました。再接続を試みています。
IVS コンポジションの状態変化	セッションの開始	コンポジションセッションが作成されました。コンポジションプロセスのパイプラインが正常に初期化されると、このイベントが発生します。この時点で、コンポジションパイプラインはステージに正常にサブスクライブされメディアを受信しており、またビデオを作成できるようになりました。
IVS コンポジションの状態変化	セッションの終了	コンポジションセッションが完了しました。
IVS コンポジションの状態変化	セッションの失敗	使用不能なステージリソースか、その他の内部エラーが原因で Composition パイプラインが初期化に失敗しました。
IVS Stage Update	Participant Published	参加者がステージへの公開を開始したとき。
IVS Stage Update	Participant Unpublished	参加者がステージへの公開を停止したとき。

Amazon IVS の Amazon EventBridge ルールを作成する

Amazon IVS が発行したイベントをトリガーするルールを作成できます。「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge でルールを作成する」にある手順に従います。サービスを選択する際に、[Interactive Video Service (IVS)] を選択します。

例: Composition の状態変化

Destination Failure: このイベントは、送信先への出力に失敗したときに送信されます。例えば、ストリームキーがないか、別のブロードキャストが進行中の場合には、あるチャンネルへのブロードキャストが失敗します。

```
{
   "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"
   }
}
```

Destination Start: このイベントは、送信先への出力が正常に開始されたときに送信されます。

```
{
   "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>",
```

```
    }
}
```

Destination End: このイベントは、送信先への出力が完了したときに送信されます。

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

Destination Reconnecting: このイベントは、送信先への出力が中断され、再接続が試行中である場合に送信されます。

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

```
}
```

Session Start: このイベントは、コンポジションセッションが作成されたときに送信されます。コンポジションプロセスのパイプラインが正常に初期化されると、このイベントが発生します。この時点で、コンポジションパイプラインはステージに正常にサブスクライブされメディアを受信しており、またビデオを作成できるようになりました。

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

Session End: このイベントは、コンポジションセッションが完了し、すべてのリソースが削除されたときに送信されます。

```
{
   "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>"
   }
```

```
}
```

Session Failure: このイベントは、ステージリソースが使用できないかステージに参加者がいない場合、またはその他の内部エラーが原因で、Composition パイプラインが初期化に失敗した場合に送信されます。

```
{
   "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"
   }
}
```

例: Stage Update

このイベントには、イベント名 (イベントを分類する) とイベントに関するメタデータが含まれます。メタデータには、イベントをトリガーした参加者 ID、関連するステージ ID とセッション ID、ユーザー ID が含まれます。

Participant Published: このイベントは、参加者がステージへの公開を開始したときに送信されます。

```
{
    "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"
    }
}
```

Participant Unpublished: このイベントは、参加者がステージへの公開を停止したときに送信されます。

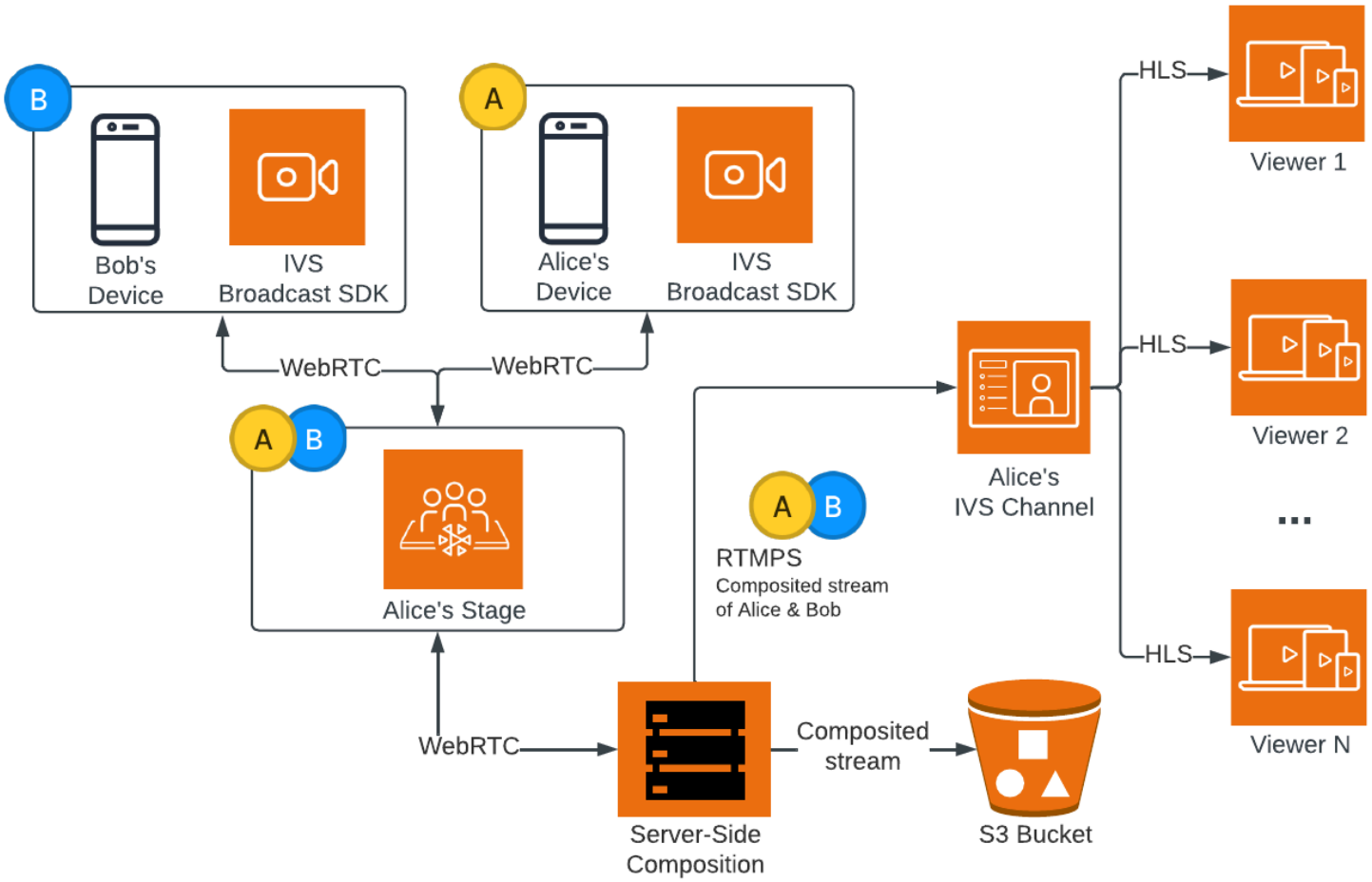
```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"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

サーバーサイドコンポジション (リアルタイムストリーミング)

サーバーサイドコンポジションでは、IVS サーバーを使用してステージ参加者全員からの音声と動画をミックスし、このミックスされたビデオを IVS チャネル (より多くの視聴者に配信する場合) または S3 バケットに送信します。サーバーサイドコンポジションは、ステージのホームリージョンにある IVS コントロールプレーンエンドポイントを介して呼び出されます。

サーバーサイドコンポジションを使用したステージのブロードキャストと録画には多くの利点があります。これは、効率的で信頼性の高いクラウドベースのビデオワークフローを求めるユーザーにとって魅力的な選択肢となっています。

この図は、サーバーサイドコンポジションの仕組みを示しています。



利点

クライアントサイドのコンポジションと比較すると、サーバーサイドコンポジションには以下の利点があります。

- クライアント負荷の軽減 – サーバーサイドコンポジションを使用すると、音声と動画のソースを処理および結合する負担が、個々のクライアントデバイスからサーバー側に移転します。サーバーサイドコンポジションにより、ビューを合成してIVS に送信するクライアントデバイスは、CPU とネットワークリソースを使用しなくても良くなります。つまり、視聴者のデバイスでは、リソースを大量に消費するタスクを処理しなくてもブロードキャストの視聴が可能になり、より長いバッテリー寿命と、よりスムーズな視聴体験が実現されます。
- 一貫した品質 – サーバーサイドコンポジションでは、最終的なストリームの品質、解像度、ビットレートを正確に制御することができます。これにより、個々のデバイスの性能に関係なく、すべての視聴者に対し一貫した視聴体験が保証されます。
- レジリエンス – コンポジションプロセスをサーバー上で一元化することで、ブロードキャストをより堅牢にできます。パブリッシャーのデバイスに技術的な制限がかかっていたり、変動が発生していたりしても、サーバーはそれに適応するので、すべての視聴者にスムーズなストリームを提供できます。
- 帯域幅の効率 – コンポジションの処理はサーバーで実行されるため、ステージパブリッシャーは、ビデオを IVS にブロードキャストする帯域幅を余分に消費する必要がありません。

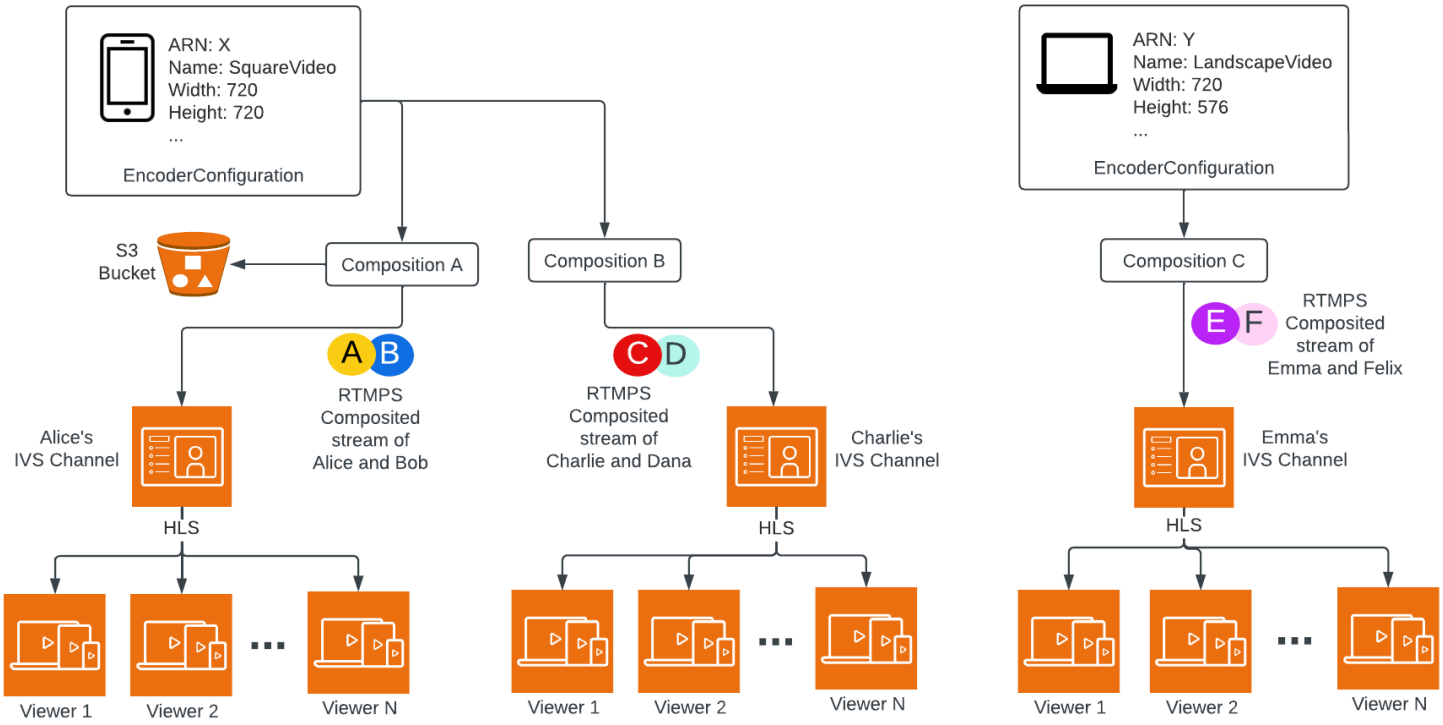
あるいは、クライアント側でコンポジションを実行し、ステージを IVS チャンネルにブロードキャストすることもできます。「IVS 低レイテンシーストリーミングユーザーガイド」の「Amazon IVS ストリームで複数ホストを有効にする」を参照してください。

IVS API

サーバーサイドコンポジションでは、主要な API 要素として以下を使用します。

- EncoderConfiguration オブジェクトは、生成する動画の形式 (高さ、幅、ビットレート、その他のストリーミングパラメータ) をカスタマイズできるようにします。EncoderConfiguration は、StartComposition エンドポイントを呼び出すたびに再利用できます。
- Composition エンドポイントはビデオコンポジションを追跡し、IVS チャンネルに出力します。
- StorageConfiguration は、コンポジションが記録されている S3 バケットを追跡します。

サーバーサイドコンポジションを使用するには、EncoderConfiguration を作成し、StartComposition エンドポイントを呼び出す際にそれをアタッチする必要があります。この例では、SquareVideo EncoderConfiguration が 2 つのコンポジションで使用されています。



詳細については、「IVS リアルタイムストリーミング API リファレンス」を参照してください。

Layouts

デフォルト設定のサーバーサイドコンポジション機能では、ステージ参加者を同じサイズのスロットに配置するために、グリッドレイアウトを使用しています。

グリッドレイアウトには、ユーザーがおすすめのスロットを設定して呼び出すオプションがあります。注目のスロットはメイン画面に表示され、その下に他の参加者が同じ大きさのスロットで表示されます。

注: サーバーサイドコンポジションのステージパブリッシャーでサポートされる最大の解像度は 1080p です。1080p を超える動画を送信するパブリッシャーは、音声のみの参加者としてレンダリングされます。

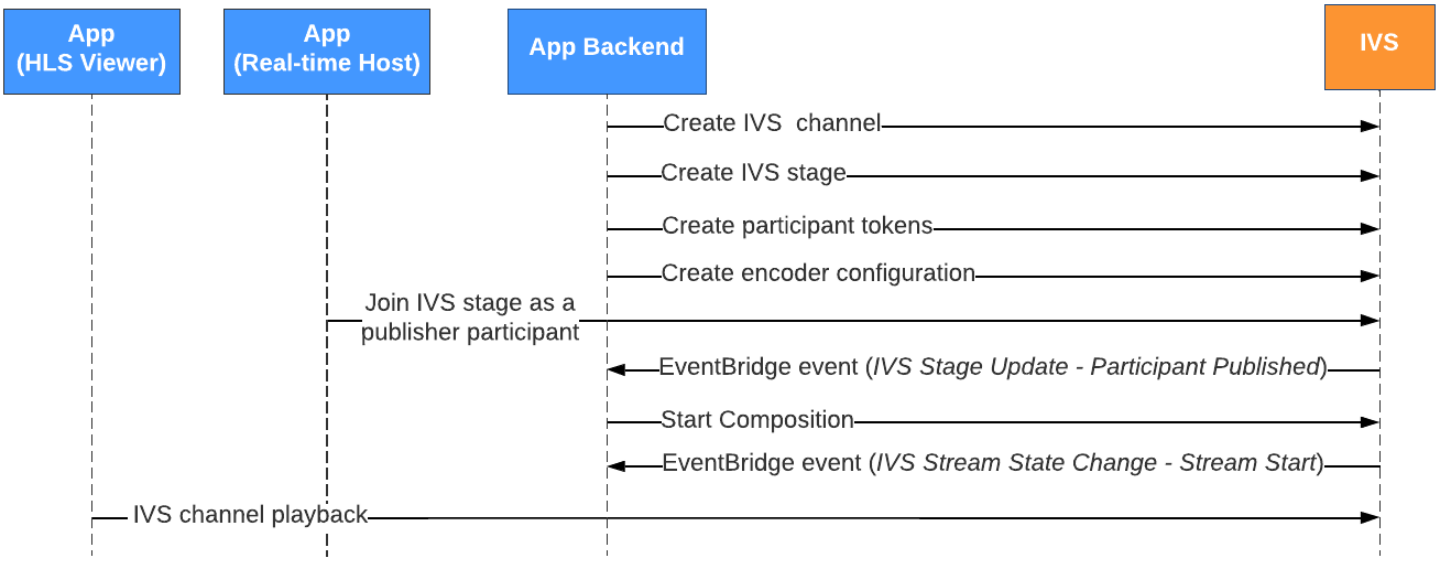
使用開始

前提条件

サーバーサイドコンポジションを使用するには、アクティブなパブリッシャーを持つステージを用意し、コンポジションの送信先として IVS チャンネルおよび (または) S3 バケットを使用する必要があります。以下では、参加者が発行した際に、ステージを IVS チャンネルにブロードキャストする

コンポジションを EventBridge のイベントにより開始する、ワークフローの 1 例について説明します。また、独自のアプリケーションロジックに基づいてコンポジションを開始および停止することもできます。S3 バケットに直接ステージを記録する、サーバーサイドコンポジションの使用方法の例については、「コンポジットの記録」を参照してください。

1. IVS チャネルを作成します。「Amazon IVS Low-Latency Streaming を開始する」を参照してください。
2. パブリッシャーごとに IVS ステージと参加者トークンを作成します。
3. EncoderConfiguration を作成します。
4. ステージに参加して公開します。(リアルタイムストリーミングブロードキャスト SDK ガイドの「公開とサブスクライブ」セクションを、以下から参照してください: Web、Android、iOS)
5. 参加者が発行した EventBridge を自分が受信した場合は、StartComposition を呼び出します。
6. 数秒待ってから、チャンネル再生で合成されたビューを確認します。



注: コンポジションは、パブリッシャーである参加者がステージ上で何も操作しない状態が 60 秒間続くと自動的にシャットダウンします。その時点でコンポジションは終了し、STOPPED 状態に移行します。STOPPED 状態に移行して数分後、コンポジションは自動的に削除されます。

CLI の手順

AWS CLI の使用は詳細オプションであり、まず CLI をダウンロードしてマシン上で設定する必要があります。詳細については、AWS Command Line Interface のユーザーガイドを参照してください。

CLI を使用してリソースを作成し、管理できるようになりました。コンポジションエンドポイントは、ivs-realtime 名前空間の下にあります。

EncoderConfiguration リソースの作成

EncoderConfiguration は、生成される動画の形式 (高さ、幅、ビットレート、その他のストリーミングパラメータ) をカスタマイズできるようにするオブジェクトです。次のステップで説明するように、EncoderConfiguration は、コンポジションエンドポイントを呼び出すたびに再利用できます。

以下のコマンドでは、動画のビットレート、フレームレート、解像度などのサーバー側のビデオコンポジションのパラメーターを設定する、EncoderConfiguration リソースを作成しています。

```
aws ivs-realtime create-encoder-configuration --name "MyEncoderConfig" --video
 "bitrate=2500000,height=720,width=1280,framerate=30"
```

レスポンスは次のとおりです。

```
{
   "encoderConfiguration": {
      "arn": "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/9W59OBY2M8s4",
      "name": "MyEncoderConfig",
      "tags": {},
      "video": {
         "bitrate": 2500000,
         "framerate": 30,
         "height": 720,
         "width": 1280
      }
   }
}
```

コンポジションの開始

上記のレスポンスで提供された EncoderConfiguration ARN を使用して、以下のコンポジションリソースを作成します。

```
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"}}]'
```

レスポンスには、STARTING 状態の Composition が作成されたことが示されます。Composition がコンポジションの発行を開始すると、状態は ACTIVE に遷移します。(この状態は、ListCompositions エンドポイントまたは GetComposition エンドポイントを呼び出すことで確認できます)。

Composition が ACTIVE になると、ListCompositions を使用して IVS ステージの合成ビューが IVS チャンネルに表示されます。

```
aws ivs-realtime list-compositions
```

レスポンスは次のとおりです。

```
{
   "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": {}
      }
   ]
}
```

注: コンポジションのアクティブな状態を維持するには、ステージへのパブリッシングを積極的に行っている (パブリッシャーである) 参加者が必要です。詳細については、リアルタイムストリーミングブロードキャスト SDK ガイドの「公開とサブスクライブ」セクションを、以下から参照してください: Web、Android、iOS。参加者ごとには、別々のステージトークンを作成する必要があります。

画面共有を有効にする

固定された画面共有レイアウトを使用するには、以下の手順を実行します。

EncoderConfiguration リソースの作成

以下のコマンドでは、サーバーサイドコンポジションのパラメーター (動画ビットレート、フレームレート、解像度) を設定する EncoderConfiguration リソースを作成しています。

```
aws ivs-realtime create-encoder-configuration --name "test-ssc-with-screen-share" --
video={bitrate=2000000,framerate=30,height=720,width=1280}
```

screen-share 属性を使用してステージ参加者トークンを作成します。screen-share は featured スロットの名前として指定しているので、screen-share 属性に true を設定したステージトークンを作成する必要があります。

```
aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes screen-share=true
```

レスポンスは次のとおりです。

```
{
   "participantToken": {
      "attributes": {
         "screen-share": "true"
      },
      "expirationTime": "2023-08-04T05:26:11+00:00",
      "participantId": "E813MFklPWLF",
      "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2OTExMjY3NzEsImlhdCI6MTY5MTA4MzU3MSwianRpIjoiRT
   }
}
```

コンポジションの開始

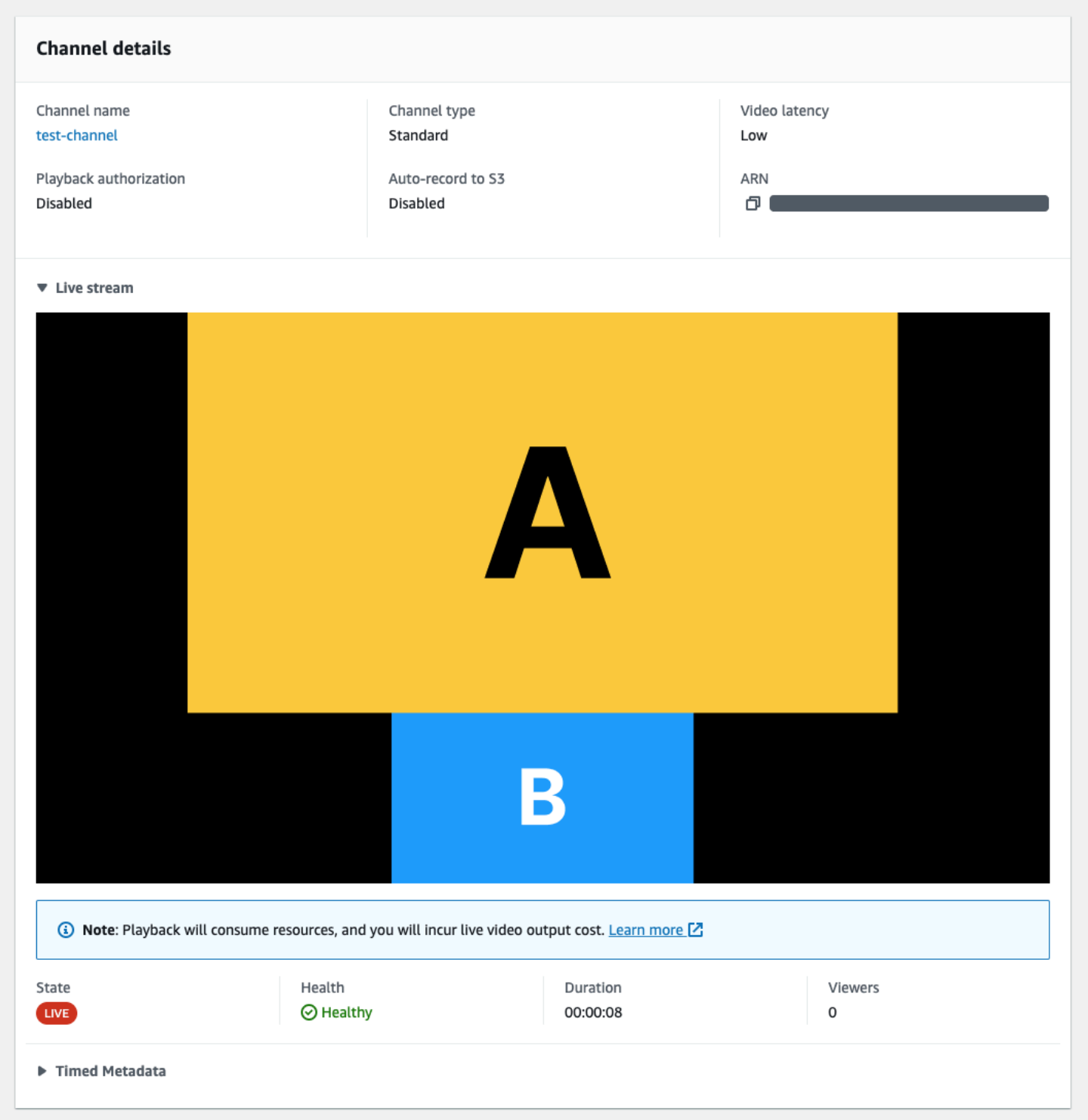
スクリーン共有機能によりコンポジションを開始するには、以下のコマンドを使用します。

```
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}"
```

レスポンスは次のとおりです。

```
{
   "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" : { }
   }
}
```

ステージ参加者 E813MFklPWLF がステージに参加すると、その参加者のビデオが、おすすめスロットに表示されます。他のすべてのステージパブリッシャーは、そのスロットの下にレンダリングされます。



Composition を停止します。

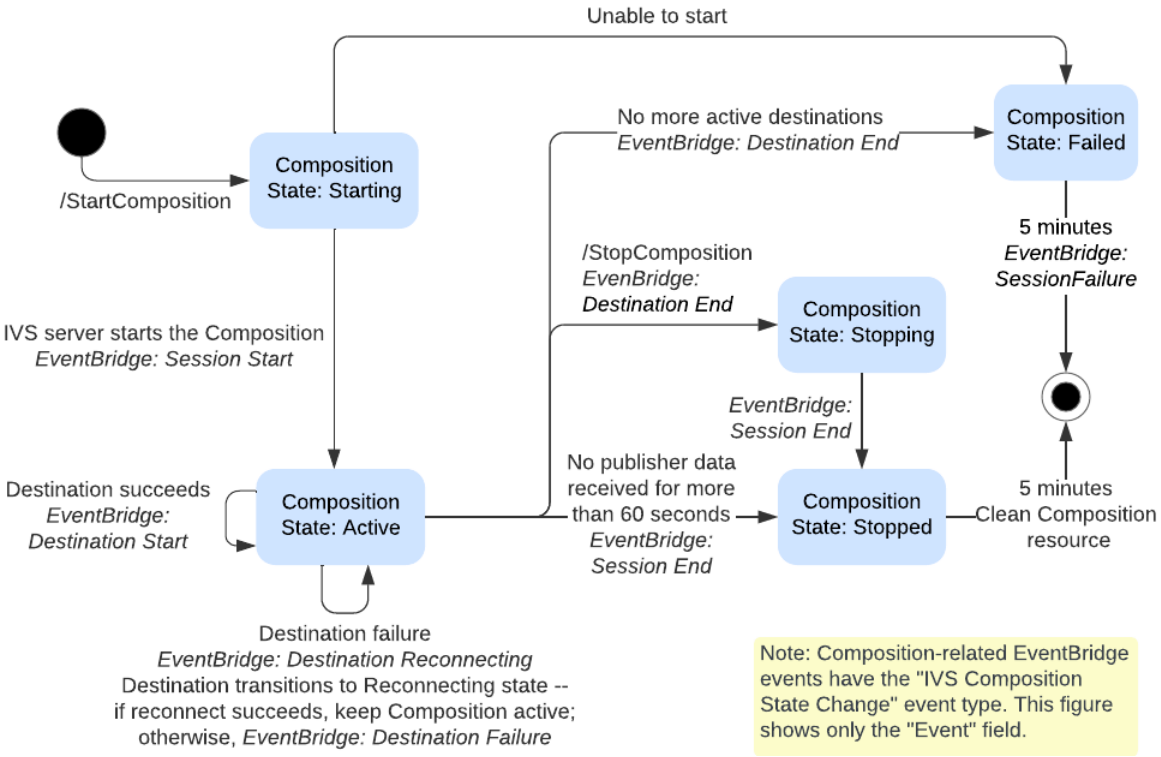
コンポジションを停止するには、いつでも、StopComposition エンドポイントを呼び出します。

```
aws ivs-realtime stop-composition --arn arn:aws:ivs:us-east-1:927810967299:composition/
B19tQcXRgtoz
```

Composition のライフサイクル

Composition の状態遷移を理解するには、下の図を参照してください。概観的には、コンポジションのライフサイクルは次のとおりです。

1. Composition リソースは、ユーザーが StartComposition エンドポイントを呼び出した際に作成されます。
2. IVS が Composition の開始に成功すると、「IVS Composition State Change (Session Start)」の EventBridge イベントが送信されます。イベントの詳細については、「IVS リアルタイムストリーミングで EventBridge を使用する」を参照してください。
3. Composition がアクティブ状態になった後は、以下のことが発生します。
   - ユーザーが Composition を停止する – StopComposition エンドポイントが呼び出されると、IVS は Composition の適切なシャットダウンを開始し、「Destination End」、「Session End」の順にイベントを送信します。
   - コンポジションが自動シャットダウンを実行 – IVS ステージにアクティブに発行している参加者いなくなってから 60 秒後に、Composition は自動的にファイナライズされ、EventBridge イベントが送信されます。
   - 送信先の障害 – 送信先で (IVS チャンネルが削除されるなどの) 予期しない障害が発生すると、その送信先は `RECONNECTING` 状態に遷移し、「Destination Reconnecting」イベントが送信されます。復旧が不可能な場合、IVS が対象の送信先を `FAILED` 状態に遷移させ、「Destination Failure」イベントが送信されます。少なくとも 1 つの送信先がアクティブであれば、IVS はコンポジションを維持します。
4. `STOPPED` あるいは `FAILED` 状態になったコンポジションは、その 5 分後に自動的にクリーンアップされます。(それ以降は、ListCompositions や GetComposition によって取得されなくなります)。


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.

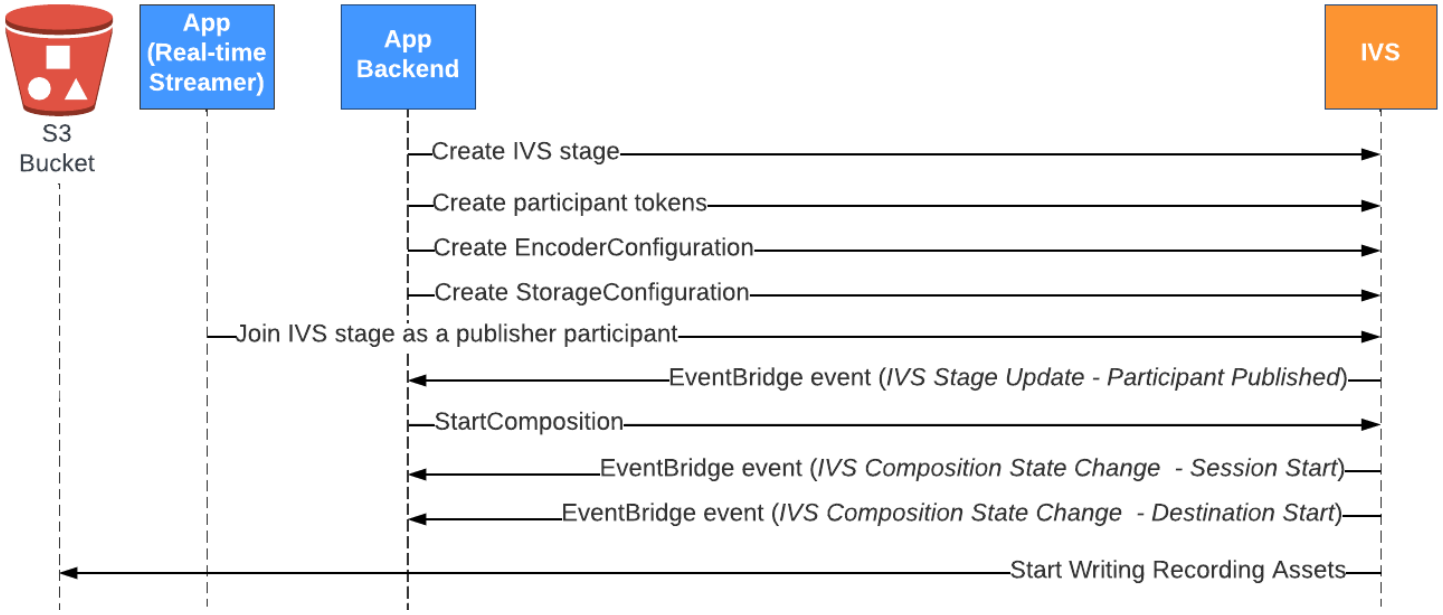
Composite Recording (リアルタイムストリーミング)

このドキュメントでは、サーバーサイドコンポジションで Composite Recording 機能を使用する方法について説明します。Composite Recording では、IVS サーバーを使用してすべてのステージパブリッシャーを 1 つのビューに効果的に結合して IVS ステージの HLS 録画を生成し、結果のビデオを S3 バケットに保存できます。

前提条件

複合録画を使用するには、アクティブなパブリッシャーを含むステージと、録画先として使用する S3 バケットが必要です。以下に、 EventBridge イベントを使用して S3 バケットへのコンポジションを記録するワークフローの 1 つについて説明します。また、独自のアプリケーションロジックに基づいてコンポジションを開始および停止することもできます。

1. パブリッシャーごとに IVS ステージと参加者トークンを作成します。
2. (録画されたビデオのレンダリング方法を表すEncoderConfigurationオブジェクト) を作成します。
3. S3 バケットと StorageConfiguration (録画コンテンツが保存される) を作成します。
4. ステージに参加して公開します。
5. Participant Published EventBridge イベントを受け取ったら、送信先として S3 DestinationConfiguration object StartCompositionを使用して を呼び出します。
6. 数秒後、HLS セグメントが S3 バケットに保持されていることがわかります。



注: コンポジションは、パブリッシャーである参加者がステージ上で何も操作しない状態が 60 秒間続くと自動的にシャットダウンします。その時点でコンポジションは終了し、STOPPED 状態に移行します。STOPPED 状態に移行して数分後、コンポジションは自動的に削除されます。詳細については、「サーバーサイドコンポジション」の「コンポジションライフサイクル」を参照してください。

複合記録の例: S3 バケット StartComposition の送信先

次の例は、S3 をコンポジションの唯一の送信先として指定する、 StartCompositionエンドポイントへの一般的な呼び出しを示しています。コンポジションが ACTIVE 状態に移行すると、storageConfiguration オブジェクトで指定された S3 バケットへのビデオセグメントとメタデータの書き込みが開始されます。異なるレイアウトのコンポジションを作成するには、「サーバーサイドコンポジション」の「レイアウト」と「IVS リアルタイムストリーミング API リファレンス」を参照してください。

リクエスト

```
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"
}
```

レスポンス

```
{
    "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": {}
    }
}
```

StartComposition レスポンスに存在する recordingPrefixフィールドを使用して、録画コンテンツの保存場所を特定できます。

録画の内容

コンポジションが ACTIVE状態に移行すると、 の呼び出し時に提供された S3 バケットに HLS ビデオセグメントとメタデータファイルが書き込まれ始めます StartComposition。これらのコンテンツは、後処理またはオンデマンド動画再生として利用できます。

コンポジションがライブになると、「IVS Composition State Change」イベントが発生し、マニフェストファイルとビデオセグメントが書き込まれるまでに少し時間がかかることに注意してください。「IVS Composition State Change (Session End)」イベントの受信後に、録画したストリームを再生または処理することをお勧めします。詳細については、「IVS リアルタイムストリーミング EventBridge での の使用」を参照してください。

以下は、ライブの IVS セッションの録画のディレクトリ構造およびコンテンツの例です。

```
MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/composite
   events
      recording-started.json
      recording-ended.json
   media
      hls
```

eventsフォルダには、録画イベントに対応するメタデータファイルが含まれています。JSON メタデータファイルは、録画が開始された時、正常に終了した時、失敗して終了した時に生成されます。

- events/recording-started.json
- events/recording-ended.json
- events/recording-failed.json

該当する events フォルダには、recording-started.json、および recording-ended.json または recording-failed.json のどちらかが含まれます。

これらには、録画されたセッションとその出力形式に関連するメタデータが含まれます。JSON の詳細を以下に示します。

media フォルダには、サポートされているメディアコンテンツが含まれています。hls サブフォルダには、コンポジションセッション中に生成されたすべてのメディアファイルとマニフェストファイルが含まれており、IVS プレーヤーで再生できます。HLS マニフェストは multivariant.m3u8 フォルダにあります。

のバケットポリシー StorageConfiguration

StorageConfiguration オブジェクトが作成されると、IVS は指定された S3 バケットにコンテンツを書き込むためのアクセス権を取得します。このアクセス権は S3 バケットのポリシーを変更することで付与されます。バケットのポリシーが IVS のアクセス権を削除するように変更されると、進行中や新規の録画はできなくなります。

以下の例は、IVS で S3 バケットに書き込めるようにする S3 バケットポリシーを示しています。

```
{
    "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"
                }
```

```
            }
        }
    ]
}
```

JSON メタデータファイル

このメタデータは JSON 形式です。これには、以下の情報が含まれています。

フィールド	Type	必須	説明
`stage_arn`	文字列	はい	コンポジションのソースとして使用されているステージの ARN。
`media`	オブジェクト	はい	この録画に使用できるメディアコンテンツの列挙型オブジェクトを含むオブジェクト。有効な値: `"hls"`。
`hls`	オブジェクト	はい	Apple HLS 形式の出力を記述する列挙型フィールド。
`duration_ms`	整数	条件付き	録画された HLS コンテンツの継続時間 (ミリ秒単位)。これは、`recording_status` が `"RECORDING_ENDED"` または `"RECORDING_ENDED_WITH_FAILURE"` のときにのみ利用できます。録画完了前に障害が発生した場合は、0 になります。
`path`	文字列	はい	HLS コンテンツが格納されている S3 プレフィックスからの相対パス。
`playlist`	文字列	はい	HLS マスタープレイリストファイルの名前。

フィールド	Type	必須	説明
renditions	オブジェクト	はい	メタデータオブジェクトのレンディション (HLS バリアント) の配列。レンディションは必ず 1 つ以上。
path	文字列	はい	このレンディションの HLS コンテンツが格納されている S3 プレフィックスからの相対パス。
playlist	文字列	はい	このレンディションのメディアプレイリストファイルの名前。
resolution_height	int	条件付き	エンコードされた動画のピクセル解像度の高さ。これは、レンディションに動画トラックが含まれている場合にのみ使用できます。
resolution_width	整数	条件付き	エンコードされた動画のピクセル解像度の幅。これは、レンディションに動画トラックが含まれている場合にのみ使用できます。

フィールド	Type	必須	説明
recording_ended_at	string	条件付き	録画終了時の RFC 3339 UTC タイムスタンプ。これは、recording_status が "RECORDING_ENDED" または "RECORDING_ENDED_WITH_FAILURE" のときにのみ利用できます。 recording_started_at とrecording_ended_at は、これらのイベントが生成されたときのタイムスタンプであり、HLS ビデオセグメントのタイムスタンプと完全に一致しない場合があります。録画時間を正確に決定するには、duration_ms フィールドを使用してください。
recording_started_at	string	条件付き	録画開始時の RFC 3339 UTC タイムスタンプ。recording_status が RECORDING_START_FAILED の場合、これは使用できません。 recording_ended_at については、上記の注意事項を参照してください。

フィールド	Type	必須	説明
recording_status	文字列	はい	録画ステータス。有効な値は、"RECORDING_STARTED" 、"RECORDING_ENDED" 、"RECORDING_START_FAILED" 、"RECORDING_ENDED_WITH_FAILURE" です。
recording_status_message	string	条件付き	ステータスの詳細情報。これは、recording_status が "RECORDING_ENDED" または "RECORDING_ENDED_WITH_FAILURE" のときにのみ利用できます。
version	文字列	はい	メタデータスキーマのバージョン。

例: 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
        }
```

```
      ]
    }
  }
}
```

例: 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
        }
      ]
    }
  }
}
```

例: 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
        }
      ]
    }
  }
}
```

プライベートバケットからの録画コンテンツの再生

デフォルトでは録画コンテンツはプライベートです。したがって、これらのオブジェクトは S3 のダイレクト URL を使用してアクセスして再生することはできません。IVS プレーヤーまたは別のプレーヤーを使用して再生するために HLS 多変量プレイリスト (m3u8 ファイル) を開こうとすると、エラー (「リクエストしたリソースへのアクセス許可がありません」という内容など) が表示されます。代わりに、これらのファイルを Amazon CloudFront CDN (コンテンツ配信ネットワーク) で再生できます。

CloudFront ディストリビューションは、プライベートバケットからコンテンツを配信するように設定できます。通常、これは、読み取りが が提供するコントロールをバイパスするオープンにアクセス可能なバケットを持つよりも望ましいです CloudFront。オリジンアクセスコントロール (OAC) を作成することで、プライベートバケットから配信されるようにディストリビューションを設定できます。OAC は、プライベートオリジンバケットに対する読み取り権限を持つ特別な CloudFront ユーザーです。OAC は、ディストリビューションを作成した後、 CloudFront コンソールまたは API を使用して作成できます。「Amazon CloudFront デベロッパーガイド」の「新しいオリジンアクセスコントロールの作成」を参照してください。

CORS を有効に CloudFront して を使用して再生をセットアップする

この例では、デベロッパーが CORS を有効にして CloudFront ディストリビューションを設定し、任意のドメインから録音を再生できるようにする方法について説明します。これは開発段階では特に役立ちますが、以下の例を本番環境のニーズに合わせて変更できます。

ステップ 1: S3 バケットを作成する

録画の保存に使用する S3 バケットを作成します。バケットは IVS ワークフローに使用するのと同じリージョンにある必要があることに注意してください。

次のように、十分なアクセス許可の CORS ポリシーをバケットに追加します。

1. AWS コンソールで、[S3 バケットアクセス許可] タブに移動します。
2. 以下の CORS ポリシーをコピーし、[クロスオリジンリソース共有 (CORS)] に貼り付けます。これにより S3 バケットの CORS アクセスが有効になります。

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

ステップ 2: CloudFront ディストリビューションを作成する

「 デベロッパーガイド」の CloudFront 「ディストリビューションの作成」を参照してください。
CloudFront

AWS コンソールを使用して、以下の情報を入力します。

このフィールドには	これを選択します
オリジンドメイン	前のステップで作成した S3 バケット
オリジンアクセス	オリジンアクセスコントロール設定 (推奨)、デフォルトパラメータを使用
デフォルトのキャッシュ動作: ビューワープロトコルポリシー	Redirect HTTP to HTTPS
デフォルトのキャッシュ動作: 許可される HTTP メソッド	GET、HEAD、OPTIONS
デフォルトのキャッシュ動作: キャッシュキーとオリジンリクエスト	CachingDisabled ポリシー
デフォルトのキャッシュ動作: オリジンリクエストポリシー	CORS-S3Origin
デフォルトのキャッシュ動作: レスポンスヘッダーポリシー	SimpleCORS
ウェブアプリケーションファイアウォール	セキュリティ保護を有効にする

次に、 CloudFront ディストリビューションを保存します。

ステップ 3: S3 バケットポリシーを設定する

1. S3 バケットに StorageConfiguration 設定した をすべて削除します。これにより、そのバケットのポリシーを作成したときに自動的に追加されたバケットポリシーがすべて削除されます。
2. CloudFront ディストリビューションに移動し、すべてのディストリビューションフィールドが前のステップで定義した状態であることを確認し、バケットポリシーをコピーします (ポリシーのコピーボタンを使用) 。
3. S3 バケットに移動します。[アクセス許可] タブで [バケットポリシーを編集] を選択し、前のステップでコピーしたバケットポリシーを貼り付けます。このステップの後、バケットポリシーには CloudFront ポリシーのみが必要です。
4. S3 バケット StorageConfigurationを指定して を作成します。

StorageConfiguration が作成されると、S3 バケットポリシーに 2 つの項目が表示されます。1 つは CloudFront によるコンテンツの読み取りを許可し、もう 1 つは IVS によるコンテンツの書き込みを許可します。 CloudFront と IVS アクセスを持つ最終バケットポリシーの例は、「例: CloudFront と IVS アクセスを持つ S3 バケットポリシー」に示されています。

ステップ 4: 録画を再生する

CloudFront ディストリビューションを正常にセットアップしてバケットポリシーを更新すると、IVS プレイヤーを使用して録音を再生できるようになります。

1. コンポジションを正常に開始し、S3 バケットに録画が保存されていることを確認します。
2. この例のステップ 1 からステップ 3 に従った後、動画ファイルは CloudFront URL から使用できます。 CloudFront URL は、Amazon CloudFront コンソールの詳細タブのディストリビューションドメイン名です。次のように表示されます。

   ```
   a1b23cdef4ghij.cloudfront.net
   ```
3. 録画した動画を CloudFront ディストリビューションで再生するには、S3 バケット で`multivariant.m3u8`ファイルのオブジェクトキーを見つけます。次のように表示されます。

   ```
   FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/
   multivariant.m3u8
   ```
4. オブジェクトキーを CloudFront URL の末尾に追加します。最終的にページ URL は次のようになります。

   ```
   https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/
   fjFKbylPb3k4/composite/media/hls/multivariant.m3u8
   ```
5. これで、最終的な URL を IVS プレーヤーのソース属性に追加して、録画全体を視聴できます。録画した動画を視聴するには、「IVS Player SDK: Web のガイド」の「使用開始」のデモを使用できます。

例: CloudFront と IVS アクセスを使用する S3 バケットポリシー

以下のスニペットは、 がプライベートバケットにコンテンツを読み取ること、IVS CloudFront がバケットにコンテンツを書き込むことを許可する S3 バケットポリシーを示しています。注: 以下のスニペットをコピーして自分のバケットに貼り付けないでください。ポリシーには、CloudFront ディストリビューションと に関連する IDs が含まれている必要があります StorageConfiguration。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CompositeWrite-7eiKaIGkC9DO",
      "Effect": "Allow",
      "Principal": {
        "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        },
        "Bool": {
          "aws:SecureTransport": "true"
        }
      }
    },
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::844311324168:distribution/
E1NG4YMW5MN25A"
        }
      }
    }
  ]
}
```

トラブルシューティング

- コンポジションが S3 バケットに書き込まれない — S3 バケットとStorageConfiguration オブジェクトが同じリージョンに作成されていることを確認します。また、バケットポリシーを確認して、IVS がバケットにアクセスできることを確認します。「 のバケットポリシーStorageConfiguration」を参照してください。
- 実行時にコンポジションが見つからない ListCompositions — コンポジションは工フェメラルリソースです。最終状態に移行すると、数分後に自動的に削除されます。
- コンポジションが自動的に停止します – ステージにパブリッシャーがいない時間が 60 秒を超えると、コンポジションは自動的に停止します。

既知の問題

Composite Recording によって書き込まれたメディアプレイリストには、コンポジションの進行中はタグ `#EXT-X-PLAYLIST-TYPE:EVENT` が付けられます。コンポジションが完了すると、タグは `#EXT-X-PLAYLIST-TYPE:VOD` に更新されます。再生をスムーズにするため、このプレイリストはコンポジションが正常に終了した後にのみ使用することをお勧めします。

OBS および WHIP サポート (リアルタイムストリーミング)

このドキュメントでは、OBS などの WHIP 互換エンコーダーを使用して IVS リアルタイムストリーミングに発行する方法について説明します。WHIP (WebRTC - HTTP Ingestion Protocol) は、WebRTC 取り込みを標準化するために開発された IETF ドラフトです。

WHIP は OBS などのソフトウェアとの互換性を可能にし、デスクトップ公開用の代替 (IVS Broadcast SDK へ) を提供します。シーントランジション、オーディオミキシング、オーバーレイグラフィックなどの高度な制作機能を備えているため、OBS に精通している上級ストリーマーには OBS が好ましいかもしれません。これにより、デベロッパーは汎用性を備えたオプションを利用できます。ブラウザを直接公開するには IVS Web Broadcast SDK を使用するか、ストリーマーはデスクトップで OBS を使用してより強力なツールを作成できます。

また、IVS Broadcast SDK の使用が実行可能でない場合や推奨されない場合にも、WHIP が役立ちます。例えば、ハードウェアエンコーダーを含むセットアップでは、IVS Broadcast SDK はオプションではない場合があります。ただし、エンコーダーが WHIP をサポートしている場合でも、エンコーダーから IVS に直接公開できます。

OBS ガイド

OBS は、バージョン 30 の WHIP をサポートしています。開始するには、OBS v30 以降をダウンロードします: https://obsproject.com/。

WHIP 経由で OBS を使用して IVS ステージに公開するには、次の手順に従います。

1. 公開機能を備えた参加者トークンを生成します。WHIP 用語では、参加者トークンはベアラートークンです。デフォルトでは、参加者トークンは 12 時間で期限切れになりますが、最大 14 日間まで延長できます。
2. [設定] をクリックします。設定パネルのストリームセクションで、サービスドロップダウンから WHIP を選択します。
3. サーバー には、https://global.whip.live-video.net/ と入力します。
4. ベアラートークン には、ステップ 2 で生成した参加者トークンを入力します。
5. ビデオ設定を通常どおりに設定しますが、いくつかの制限があります。
   a. IVS リアルタイムストリーミングは、8.5 Mbps で最大 720p の入力をサポートします。これらの制限のいずれかを超えると、ストリームは切断されます。

b. 出力パネルのキーフレーム間隔を 1 秒または 2 秒に設定することをお勧めします。キーフレーム間隔を短くすると、視聴者の動画再生をより迅速に開始できます。また、CPU 使用率プリセットを Ultrafast に設定し、チューニングをゼロレイテンシーに設定して、レイテンシーを最小にすることをお勧めします。

c. OBS はサイマルキャストをサポートしていないため、ビットレートは 2.5 Mbps 未満にしておくことをお勧めします。これにより、低帯域幅接続のビューワーを監視できます。

6. ストリーミングの開始 を押します。

Service Quotas (リアルタイムストリーミング)

Amazon Interactive Video Service (IVS) のリアルタイムエンドポイント、リソース、およびその他のオペレーションのサービスクオータと制限は以下のとおりです。サービスクォータ (制限とも呼ばれます) とは、AWS アカウントのサービスリソースまたはオペレーションの最大数のことです。つまりは、これらの制限は、以下の表に明記されていない限り AWS のアカウントごとに適用されます。AWS Service Quotas も参照してください。

AWS サービスにプログラムで接続するには、エンドポイントを使用します。AWS サービスエンドポイントも参照してください。

すべてのクォータはリージョンごとに適用されます。

サービスクォータの引き上げ

調整可能なクォータについては、AWS コンソールからレートの増加をリクエストできます。コンソールでは、サービスクォータに関する情報も閲覧できます。

API コールレートクォータは調整できません。

API コールレートクォータ

エンドポイントタイプ	エンドポイント	デフォルト値
コンポジション	`GetComposition`	5 TPS
コンポジション	`ListCompositions`	5 TPS
コンポジション	`StartComposition`	5 TPS
コンポジション	`StopComposition`	5 TPS
MediaEncoder	`CreateEncoderConfiguration`	5 TPS
MediaEncoder	`DeleteEncoderConfiguration`	5 TPS
MediaEncoder	`GetEncoderConfiguration`	5 TPS
MediaEncoder	`ListEncoderConfigurations`	5 TPS

エンドポイントタイプ	エンドポイント	デフォルト値
ステージ	CreateParticipantToken	50 TPS
ステージ	CreateStage	5 TPS
ステージ	DeleteStage	5 TPS
ステージ	DisconnectParticipant	5 TPS
ステージ	GetParticipant	5 TPS
ステージ	GetStage	5 TPS
ステージ	GetStageSession	5 TPS
ステージ	ListStages	5 TPS
ステージ	UpdateStage	5 TPS
ステージ	ListParticipants	5 TPS
ステージ	ListParticipantEvents	5 TPS
ステージ	ListStageSessions	5 TPS
StorageConfiguration	CreateStorageConfiguration	5 TPS
StorageConfiguration	DeleteStorageConfiguration	5 TPS
StorageConfiguration	GetStorageConfiguration	5 TPS
StorageConfiguration	ListStorageConfigurations	5 TPS
タグ	ListTagsForResource	10 TPS
タグ	TagResource	10 TPS
タグ	UntagResource	10 TPS

その他のクォータ

リソースまたは機能	デフォルト値	引き上げ可能	説明
EncoderConfigurations	20	はい	アカウントあたりのエンコーダー設定リソースの最大数。
コンポジションの送信先	2	いいえ	コンポジションリソース内の Destination オブジェクトの最大数。
コンポジション: 最大持続時間	24	いいえ	コンポジションが存続できる最大時間 (時間単位)。
コンポジション	5	はい	アカウントあたりの同時コンポジションリソースの最大数。
参加者の公開期間またはサブスクライブ期間	24	いいえ	参加者がステージを公開またはサブスクライブし続けることができる最大時間 (時間単位)。
参加者が公開する解像度	720p	いいえ	参加者が公開する動画の最大解像度。
参加者のダウンロードビットレート	8.5 Mbps	いいえ	参加者のすべてのサブスクリプションにおける、最大合計ダウンロードビットレート。
ステージ参加者 (パブリッシャー)	12	いいえ	一度に 1 つのステージに公開できる参加者の最大数。
ステージ参加者 (サブスクライバー)	10,000	はい	一度に 1 つのステージにサブスクライブできる参加者の最大数。
ステージ	100	はい	AWS リージョンあたりのステージの最大数。

リアルタイムストリーミングの最適化

ユーザーが IVS リアルタイムストリーミングを使用して動画をストリーミングおよび視聴するときに、最高のエクスペリエンスを得られるように、現在提供されている機能を使用して、エクスペリエンスの一部を向上/最適化する方法が何通りかあります。

序章

ユーザーエクスペリエンスの質を最適化するには、ユーザーが希望するエクスペリエンスを考慮することが重要です。このようなエクスペリエンスは、視聴するコンテンツやネットワークの状態によって異なる可能性があります。

このガイドでは、ストリームのパブリッシャーまたはサブスクライバーであるユーザーに焦点を当て、それらのユーザーに望ましいアクションとエクスペリエンスを考慮します。

アダプティブストリーミング: サイマルキャストによるレイヤードエンコーディング

この機能は、以下のクライアントバージョンでのみサポートされています。

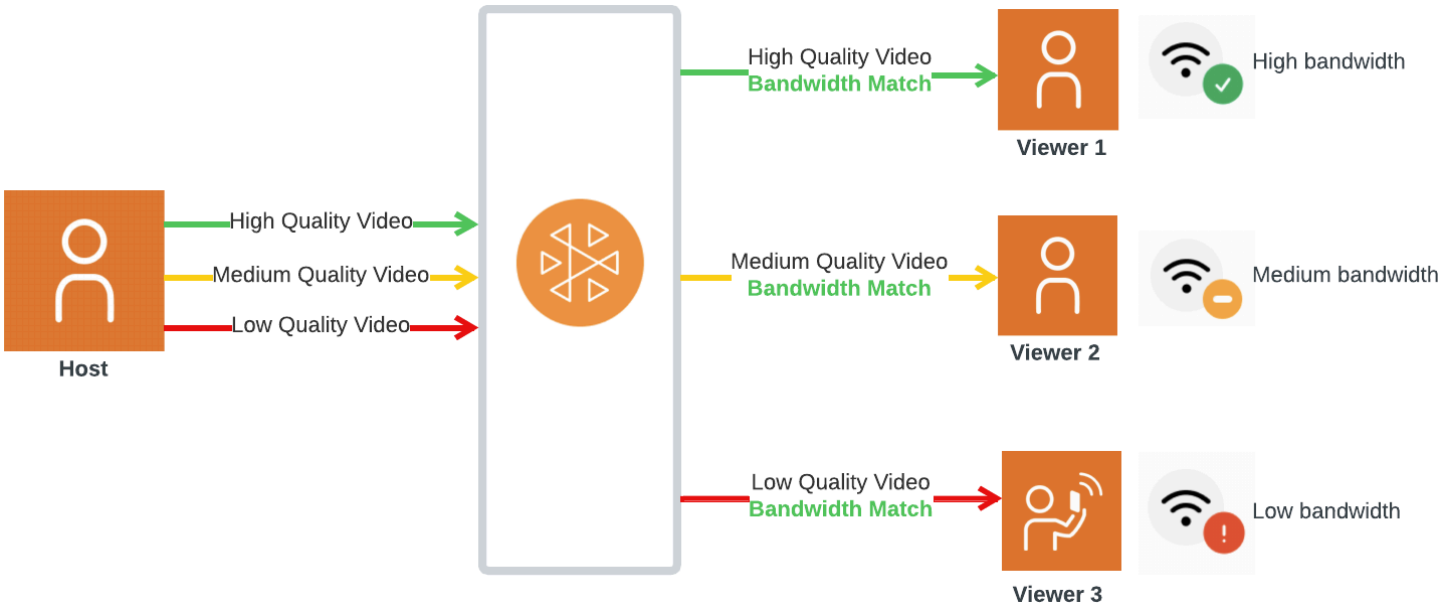
- iOS とAndroid 1.12.0+
- ウェブ 1.5.1 +

アカウントのこの機能にオプトインするには、amazon-ivs-simulcast@amazon.com まで E メールで送信する必要があります。オプトインしていない限り、SDK 設定を介してサイマルキャストを有効にしても効果はありません。

IVS のリアルタイムブロードキャスト SDK を使用する場合で、この機能にオプトインすると、パブリッシャーは複数のレイヤーのビデオをエンコードし、サブスクライバーは自動的に、ネットワークに最適な品質に適応または変更するようになります。これをサイマルキャストによるレイヤードエンコーディングと呼びます。

サイマルキャストによるレイヤードエンコーディングは Android と iOS、および Chrome デスクトップブラウザ (Windows と macOS 用) でサポートされています。他のブラウザではレイヤードエンコーディングはサポートされていません。

下の図では、ホストは 3 つのビデオ品質 (高、中、低) を送信しています。IVS は、利用可能な帯域幅に基づいて最高品質のビデオを各視聴者に転送し、各視聴者に最適な体験を提供します。Viewer 1 のネットワーク接続が正常から不良に変わると、IVS は自動的に低画質のビデオを Viewer 1 に送信し始めるため、Viewer 1 はストリームを中断されることなく (可能な限り最高の品質で) 視聴し続けることができます。



デフォルトのレイヤー、画質、フレームレート

モバイルユーザーと Web ユーザーに提供されるデフォルトの画質とレイヤーは次のとおりです。

モバイル (Android、iOS)	Web (Chrome)
高レイヤー (またはカスタム): • 最大ビットレート: 900,000 bps • フレームレート: 15 fps • 解像度: 360x640	高レイヤー (またはカスタム): • 最大ビットレート: 1,700,000 bps • フレームレート: 30 fps • 解像度: 1280x720
中間レイヤー: なし (モバイルでは高レイヤーと低レイヤーのビットレートの差が小さいため不要)	中間レイヤー: • 最大ビットレート: 700,000 bps • フレームレート: 20 fps • 解像度: 640 x 360
低レイヤー:	低レイヤー:

モバイル (Android、iOS)	Web (Chrome)
• 最大ビットレート: 150,000 bps • フレームレート: 15 fps • 解像度: 180x320	• 最大ビットレート: 200,000 bps • フレームレート: 15 fps • 解像度: 320x180

サイマルキャストによるレイヤードエンコーディングの設定

サイマルキャストでレイヤードエンコーディングを使用するには、機能をオプトインし、クライアントで有効にしておく必要があります。有効にすると、送信ビットレート全体が向上し、ビデオのフリーズが少なくなるという利点があります。

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
```

ストリーミング設定

このセクションでは、ビデオストリームとオーディオストリームで行うことができるその他の設定について説明します。

ビデオストリームビットレートの変更

ビデオストリームのビットレートを変更するには、次の設定サンプルを使用します。

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
```

ビデオストリームフレームレートの変更

ビデオストリームのフレームレートを変更するには、次の設定サンプルを使用します。

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
```

オーディオビットレートとステレオサポートの最適化

オーディオストリームのビットレートとステレオ設定を変更するには、次の設定サンプルを使用します。

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
```

推奨される最適化

シナリオ	レコメンデーション
テキスト、またはプレゼンテーションやスライドなどの動きの遅いコンテンツを含むストリーミング	サイマルキャストによるレイヤードエンコーディングを使用するか、より低いフレームレートでストリームを構成します。
アクションや多くの動きを含むストリーム	サイマルキャストによるレイヤードエンコーディングを使用します。
会話や、とても少ない動きを含むストリーム	サイマルキャストによるレイヤードエンコーディングを使用するか、音声のみを選択します (リアルタイムストリーミング Broadcast SDK ガイドの「参加者への登録」を以下から参照してください：Web、Android、iOS)。

シナリオ	レコメンデーション
限られたデータでストリーミングを行うユーザー	サイマルキャストによるレイヤードエンコーディングを使用するか、全員のデータ使用量を抑えたい場合は、フレームレートを低く設定してビットレートを手動で下げてください。

リソースおよびサポート (リアルタイムストリーミング)

リソース

https://ivs.rocks/ は、公開済みコンテンツ (デモ、コードサンプル、ブログ投稿) を閲覧し、コストを見積もり、ライブデモを通じて Amazon IVS を体験するための専用サイトです。

デモ

iOS および Android 向けの IVS リアルタイムストリーミングデモでは、開発者向けに、Amazon IVS を使用してソーシャルユーザーが生成する魅力的なリアルタイムコンテンツアプリケーションを構築する方法を紹介します。このアプリケーションは、ユーザーが生成したリアルタイムストリームのスクロール可能なフィードを扱います。ユーザーはビデオストリームとオーディオのみのルームを作成できます。ビデオストリームのゲストは、ゲストスポットまたは対 (VS) モードで参加できます。必要なバックエンドをデプロイしてアプリケーションをビルドする方法については、次の GitHub リポジトリを参照してください。

- 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/
- バックエンド: https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

サポート

AWS サポートセンターでは、AWS ソリューションをサポートするツールと専門知識へのアクセスを提供する一連のプランを利用できます。すべてのサポートプランで、24 時間年中無休のカスタマーサービスをご利用いただけます。AWS 環境の計画、デプロイ、最適化のために技術サポートや追加のリソースが必要な場合は、お客様の AWS ユースケースに合ったサポートプランを選択してください。

AWS プレミアムサポートは、1 対 1 での迅速な対応を行うサポートチャネルであり、AWS でのアプリケーションの構築と運用を支援します。

AWS re:Post は、Amazon IVS に関連する技術的な質問についてディスカッションするコミュニティベースの開発者向け Q&A サイトです。

お問い合わせ - 請求やアカウントに関する非技術的なお問い合わせ用のリンクがあります。技術的な質問の場合は、上記のディスカッションフォーラムまたはサポートリンクをご利用ください。

用語集

「AWS 用語集」も参照してください。下の表では、LL は IVS 低レイテンシーストリーミング、RT、IVS リアルタイムストリーミングを表します。

言葉	説明	LL	RT	チャット
AAC	高度なオーディオコーディング。AAC は、非可逆デジタルオーディオ圧縮用のオーディオコーディング規格です。MP3 形式の後継として設計された AAC は、通常、同じビットレートで MP3 よりも高い音質を実現します。AAC は MPEG–2 および MPEG–4 の仕様の一部として ISO と IEC によって標準化されています。	✓	✓	
アダプティブビットレートのストリーミング	アダプティブビットレート (ABR) ストリーミングにより、IVS プレーヤーは接続品質が低下した場合は低いビットレートに切り替え、接続品質が向上した場合は高いビットレートに戻すことができます。	✓		
アダプティブストリーミング	「サイマルキャストによるレイヤードエンコーディング」を参照してください。		✓	
管理ユーザー。	AWS アカウントで利用可能なリソースとサービスへの管理アクセス権を持つ AWS ユーザー。「AWS セットアップユーザーガイド」の「用語」を参照してください。	✓	✓	✓
ARN	AWS リソースに固有の識別子である Amazon リソースネーム。具体的な ARN 形式は、リソースの種類によって異なります。IVS リソースで使用される ARN 形式については、「サービス認証リファレンス」を参照してください。	✓	✓	✓

言葉	説明	LL	RT	チャット
アスペクト比	フレーム幅とフレーム高さの比率について説明します。たとえば、16:9 はフル HD または 1080p の解像度に対応するアスペクト比です。	✓	✓	
オーディオモード	さまざまなタイプのモバイルデバイスユーザーや使用する機器に合わせて最適化された、プリセットまたはカスタムのオーディオ設定。「IVS Broadcast SDK: モバイルオーディオモード (リアルタイムストリーミング)」を参照してください。		✓	
AVC、H.264、MPEG–4 Part 10	アドバンストビデオコーディング (H.264 または MPEG–4 Part 10 とも呼ばれる) は、非可逆デジタルビデオ圧縮用のビデオ圧縮規格です。	✓	✓	
背景置換	ライブストリームのクリエイターが背景を変更できるようにするカメラフィルターの一種。「IVS Broadcast SDK: サードパーティーのカメラフィルター (リアルタイムストリーミング)」の「背景の置換」を参照してください。		✓	
ビットレート	1 秒あたりに送受信されるビット数のストリーミングメトリック。	✓	✓	
ブロードキャスト、配信者	ストリーム、ストリーマーのためのその他の用語。	✓		
バッファリング	コンテンツが再生されることになっている時点よりも前に再生デバイスがコンテンツをダウンロードできない場合に発生する状態。バッファリングは、コンテンツがランダムに停止および開始する (「途切れ」とも呼ばれます)、コンテンツが長時間にわたって停止する (「フリーズ」とも呼ばれます)、または IVS プレイヤーが再生を一時停止するなど、いくつかの態様で現れる可能性があります。	✓	✓	

言葉	説明	LL	RT	チャット
バイト範囲プレイリスト	標準の HLS プレイリストよりも詳細なプレイリスト。標準 HLS プレイリストは 10 秒のメディアファイルで構成されています。バイト範囲のプレイリストでは、セグメント再生時間はストリームに設定されたキーフレーム間隔と同じです。 バイトレンジプレイリストは、S3 バケットに自動記録されたブロードキャストでのみ使用できます。HLS プレイリストに加えて作成されます。「Amazon S3 への自動レコーディング (低レイテンシーストリーミング)」の「バイトレンジプレイリスト」を参照してください。	✓		
CBR	コンスタントビットレートとは、ブロードキャスト中に起こる事象に関わらず、動画の再生中ずっと一定のビットレートを維持するエンコーダー用のレート制御方法です。アクション中の小康状態は希望のビットレートになるようにパディングされ、ピークはターゲットビットレートに合うようにエンコーディングの品質を調整することで量子化できます。VBR ではなく CBR を使用することを強くお勧めします。	✓	✓	
CDN	コンテンツ配信ネットワーク (Content Delivery Network または Content Distribution Network) は、ストリーミングビデオなどのコンテンツをユーザーのいる場所に近づけることで配信を最適化する、地理的に分散したソリューションです。	✓		

言葉	説明	LL	RT	チャット
チャンネル	インジェストサーバー、ストリームキー、再生URL、録画オプションなど、ストリーミングの設定を保存する IVS リソース。ストリーマーは、チャネルに関連付けられたストリームキーを使用してブロードキャストを開始します。すべてのメトリクスとブロードキャスト中に生成されるイベントは、チャネルリソースに関連付けられています。	✓		
チャネルタイプ	チャネルの許容解像度とフレームレートを決定します。「IVS 低レイテンシーストリーミング API リファレンス」の「チャネルタイプ」を参照してください。	✓		
チャットのログ記録	ログ記録設定をチャットルームに関連付けることで有効にできる詳細オプション。			✓
チャットルーム	メッセージレビューハンドラーやチャットロギングなどのオプション機能を含む、チャットセッションの設定を保存する IVS リソース。「IVS Chat の開始方法」の「ステップ 2: チャットルームを作成する」を参照してください。			✓
クライアントサイドコンポジション	ホストデバイスを使用してステージ参加者からのオーディオストリームとビデオストリームをミックスし、それらをコンポジットストリームとして IVS チャネルに送信します。これにより、クライアントリソースの使用率が高くなり、ステージやホストの問題が視聴者に影響を与えるリスクは高まりますが、コンポジションの外観をより細かく制御できます。 「サーバーサイドコンポジション」も参照してください。	✓	✓	

言葉	説明	LL	RT	チャット
CloudFront	Amazon が提供する CDN サービス。	✓		
CloudTrail	AWS や外部ソースからのイベントやアカウントアクティビティを収集、監視、分析、保持するための AWS サービス。「AWS での IVS API コールのログ記録 CloudTrail」を参照してください。	✓	✓	✓
CloudWatch	アプリケーションの監視、パフォーマンスの変化への対応、リソース使用の最適化、および運用状況に関するインサイトの提供を行う AWS サービス。 CloudWatch を使用して IVS メトリクスをモニタリングできます。「IVS リアルタイムストリーミングのモニタリング」および「IVS 低レイテンシーストリーミングのモニタリング」を参照してください。	✓	✓	✓
コンポジション	複数のソースからのオーディオストリームとビデオストリームを 1 つのストリームにまとめるプロセス。	✓	✓	
コンポジションパイプライン	複数のストリームを結合し、結果のストリームをエンコードするために必要な一連の処理ステップ。	✓	✓	
圧縮	元の表示よりも少ないビット数で情報をエンコードします。いずれの圧縮も、可逆圧縮または非可逆圧縮です。可逆圧縮は、統計上の冗長性を特定して排除することでビット数を削減します。可逆圧縮では情報が失われることはありません。非可逆圧縮は、不要な情報や重要度の低い情報を削除することでビット数を削減します。	✓	✓	

言葉	説明	LL	RT	チャット
コントロールプレーン	チャネル、ステージ、チャットルームなどの IVS リソースに関する情報を保存し、これらのリソースを作成および管理するためのインターフェースを提供します。リージョナルであり、AWS リージョンに基づきます。	✓	✓	✓
CORS	Cross-Origin Resource Sharing は、特定のドメインにロードされたクライアントウェブアプリケーションが異なるドメイン内 S3 バケットなどのリソースと通信する方法を定義します。アクセスはヘッダー、HTTP メソッド、オリジンドメインに基づいて設定できます。「Amazon Simple Storage Service ユーザーガイド」の「クロスオリジンリソース共有 (CORS) の使用 – Amazon Simple Storage Service」を参照してください。	✓		
カスタム画像ソース	IVS Broadcast SDK が提供するインターフェイス。プリセットカメラに限定されず、アプリケーションが独自の画像入力を行えるようにします。	✓	✓	
データプレーン	データを取り込みから出力まで伝送するインフラストラクチャ。コントロールプレーンで管理される設定に基づいて動作し、AWS リージョンに限定されません。	✓	✓	✓
エンコーダ。エンコーディングします	動画やオーディオコンテンツをストリーミングに適したデジタル形式に変換するプロセス。エンコーディングはハードウェアベースでもソフトウェアベースでもかまいません。	✓	✓	

言葉	説明	LL	RT	チャット
イベント	IVS がモニタリングサービスに発行する AmazonEventBridge自動通知。イベントは、ステージやコンポジションパイプラインなどのストリーミングリソースの状態や状態の変化を表します。「Amazon EventBridge と IVS 低レイテンシーストリーミングの使用」および「Amazon EventBridge と IVS リアルタイムストリーミングの使用」を参照してください。	✓	✓	✓
FFmpeg	動画やオーディオのファイルやストリームを処理するためのライブラリとプログラム群で構成される、無料でオープンソースのソフトウェアプロジェクト。FFmpeg は、オーディオと動画を録画、変換、ストリーミングするためのクロスプラットフォームソリューションを提供します。	✓		
断片化されたストリーム	ブロードキャストが切断され、チャネルの録画設定で指定された時間内に再接続されるときに作成されます。生成された複数のストリームは、単一のブロードキャストと見なされ、マージされ単一の録画ストリームになります。「Amazon S3 への自動記録(低レイテンシーストリーミング)」の「フラグメント化されたストリームのマージ」を参照してください。	✓		
フレームレート	1 秒あたりに送受信される動画フレーム数のストリーミングメトリック。	✓	✓	
HLS	HTTP ライブストリーミング (HLS) は、IVS ストリームを視聴者に配信するために使用される HTTP ベースのアダプティブビットレートストリーミング通信プロトコルです。	✓		

言葉	説明	LL	RT	チャット
HLS のプレイリスト	ストリームを構成するメディアセグメントのリスト。標準 HLS プレイリストは 10 秒のメディアファイルで構成されています。HLS は、より詳細なバイト範囲のプレイリストをサポートしています。	✓		
ホスト	ビデオやオーディオをステージに送信するリアルタイムのイベント参加者。		✓	
IAM	Identity and Access Management は、ユーザーが ID を管理し、IVS を含む AWS のサービスとリソースへのアクセスを安全に管理できるようにする AWS サービスです。	✓	✓	✓
取り込み	IVS は、ホストまたはブロードキャストから動画ストリームを受信して処理または視聴者や他の参加者に配信するためのプロセスです。	✓	✓	
取り込みサーバー	ビデオストリームを受信してトランスコーディングシステムに配信します。トランスコーディングシステムでは、ストリームがトランスミックスされるか、HLS にトランスコーディングされ、視聴者に配信されます。 インジェストサーバーは、取り込みプロトコル (RTMP、RTMPS) とともにチャネルのストリームを受信する特定の IVS コンポーネントです。チャネルの作成については、「IVS 低レイテンシーストリーミングの開始」を参照してください。		✓	
インターレースビデオ	後続のフレームの奇数行または偶数行のみを送信して表示し、余分な帯域幅を消費せずに、体感フレームレートを倍増させます。ビデオ品質の問題から、インターレースビデオの使用はお勧めしません。	✓	✓	

言葉	説明	LL	RT	チャット
JSON	JavaScript Object Notation は、人間が読み取れるテキストを使用して、属性と値のペア、配列データ型、またはその他の直列化可能な値で構成されるデータオブジェクトを送信するオープンスタンダードのファイル形式です。	✓	✓	✓
キーフレーム、デルタフレーム、キーフレーム間隔	キーフレーム (イントラコードまたは i フレームとも呼ばれます) は、動画内の画像のフルフレームです。後続のフレームであるデルタフレーム (予測フレームまたは P フレームとも呼ばれます) には、変更された情報のみが含まれます。キーフレームは、エンコーダーで定義されているキーフレーム間隔に応じて、ストリーム内に複数回表示されます。	✓	✓	
Lambda	サーバーインフラストラクチャをプロビジョニングせずにコード (Lambda 関数と呼ばれる) を実行するための AWS サービス。Lambda 関数は、イベントや呼び出しリクエストに応答して実行することも、スケジュールに基づいて実行することもできます。たとえば、IVS Chat は Lambda 関数を使用してチャットルームのメッセージレビューを有効にします。	✓	✓	✓
レイテンシー、glass-to-glassレイテンシー	データ転送の遅延。IVS はレイテンシー範囲を次のように定義しています。 • 低レイテンシー: 3 秒未満 • リアルタイムレイテンシー: 300 ms 未満 Glass-to-glass レイテンシーとは、カメラがライブストリームをキャプチャしてから視聴者の画面に表示されるまでの遅延を指します。	✓	✓	

言葉	説明	LL	RT	チャット
サイマルキャストによるレイヤードエンコーディング。	品質レベルの異なる複数のビデオストリームを同時にエンコードして公開できます。「リアルタイムストリーミングによる最適化」の「アダプティブストリーミング: サイマルキャストによるレイヤードエンコーディング」。		✓	
メッセージレビューハンドラ	IVS Chat の顧客に、チャットルームに配信される前にユーザーチャットメッセージを自動的に確認/フィルタリングする機能を付与します。Lambda 関数をチャットルームに関連付けることで有効になります。「チャットメッセージレビューハンドラー」の「Lambda 関数の作成」を参照してください。			✓
ミキサー	IVS Mobile Broadcast SDK の機能の 1 つとして、複数のオーディオおよびビデオソースを受け取り、単一の出力を生成します。カメラ、マイク、スクリーンキャプチャ、アプリケーションで生成されたオーディオと動画などのソースを表す画面上のビデオとオーディオ要素の管理をサポートします。その後、出力を IVS にストリーミングできます。「IVS Broadcast SDK: Mixer ガイド (低レイテンシーストリーミング)」の「ミキシング用のブロードキャストセッションの設定」を参照してください。	✓		

言葉	説明	LL	RT	チャット
マルチホストストリーミング	複数のホストからのストリームを 1 つのストリームにまとめます。これは、クライアント側またはサーバー側のコンポジションを使用して実現できます。 マルチホストストリーミングでは、視聴者をステージに招待して質疑応答、ホスト同士の競争、ビデオチャット、大勢の視聴者の前でホスト同士が会話するなどのシナリオが可能になります。		✓	
マルチバリアントプレイリスト	ブロードキャストで利用できるすべてのバリアントストリームのインデックス。	✓		
OAC	S3 バケットへのアクセスを制限し、録画されたストリームなどのコンテンツを CloudFront CDN 経由でのみ提供できるようにするメカニズムであるオリジンアクセスコントロール。	✓		
OBS	オープンブロードキャスタソフトウェア (OBS) – 動画の録画と ライブストリーミング用のオープンソースの無料ソフトウェア。OBS はデスクトップパブリッシング用の (IVS ブロードキャスト SDK に代わる) 代替手段を提供します。シーントランジション、オーディオミキシング、オーバーレイグラフィックなどの高度な制作機能を備えているため、OBS に精通している上級ストリーマーには OBS が好ましいかもしれません。	✓	✓	
Participant	ホストまたは視聴者としてステージに接続したリアルタイムのユーザー。		✓	
参加者トークン	イベント参加者がステージに参加すると、その参加者をリアルタイムで認証します。参加者トークンは、参加者がステージに動画を送信できるかどうかも制御します。		✓	

言葉	説明	LL	RT	チャット
再生トークン、再生キーペア	顧客がプライベートチャネルでの動画再生を制限できるようにする認証メカニズム。再生トークンは、再生 キーペアから生成されます。 再生キーペアは、再生用の視聴者認証トークンに署名して検証するために使用されるパブリッククキーとプライベートキーのペアです。「プライベートチャネルの設定」の「再生キーの作成またはインポート」と、「IVS 低レイテンシー API リファレンス」の「再生キーペアのエンドポイント」を参照してください。	✓		
再生 URL	視聴者が特定のチャネルの再生を開始するために使用するアドレスを識別します。このアドレスはグローバルに使用できます。IVS は、IVS グローバルコンテンツ配信ネットワーク上で最適な場所を自動的に選択し、各視聴者に動画を配信します。チャネルの作成については、「IVS 低レイテンシーストリーミングの開始」を参照してください。	✓		
￥プライベートチャネル	再生トークンに基づく認証メカニズムを使用して、お客様がストリームへのアクセスを制限できるようにします。「プライベートチャネルの設定」の「プライベートチャネルのワークフロー」を参照してください。	✓		
プログレッシブビデオ	各フレームのすべての行を順番に送信して表示します。ブロードキャストのすべての段階でプログレッシブビデオを使用することをお勧めします。	✓	✓	

言葉	説明	LL	RT	チャット
クォータ	AWS アカウントの IVS サービスリソースまたはオペレーションの最大数。つまり、これらの制限は、特に断りのない限り、AWS のアカウントごとに適用されます。すべてのクォータはリージョンごとに適用されます。「AWS 一般リファレンスガイド」の「Amazon インタラクティブビデオサービスのエンドポイントとクォータ」を参照してください。	✓	✓	✓
リージョン	リージョンを使用すると、特定の地域に物理的に存在する AWS のサービスにアクセスすることができます。リージョンでは耐障害性や安定性が提供され、レイテンシーを低減することもできます。これにより、リージョンの障害の影響を受けずに利用できる冗長リソースを作成できます。 ほとんどの AWS サービスのリクエストは特定の地域に関するものです。あるリージョンで作成したリソースは、AWS サービスで提供されるレプリケーション機能を明示的に使用しないかぎり、他のリージョンに存在することはありません。たとえば、Amazon S3 はクロスリージョンのレプリケーションをサポートしています。IAM などの一部のサービスには、リージョン間のリソースがありません。	✓	✓	✓
解決方法	1 つの動画フレームのピクセル数を示します。たとえば、フル HD または 1080p は 1920 x 1080 ピクセルのフレームを定義します。	✓	✓	
ルートユーザー	AWS アカウントの所有者。ルートユーザーは、AWS アカウントのすべての AWS サービスとリソースへの完全なアクセス権を持ちます。	✓	✓	✓

言葉	説明	LL	RT	チャット
RTMP、RTMPS	Real–Time Messaging Protocol。ネットワーク上でオーディオ、動画、データを送信するための業界標準。RTMPS は、Transport Layer Security (TLS/SSL) 接続上で実行される RTMP の安全なバージョンです。	✓	✓	
S3 バケット	Amazon S3 に格納されたオブジェクトのコレクション。アクセスやレプリケーションを含む多くのポリシーがバケットレベルで定義され、バケット内のすべてのオブジェクトに適用されます。たとえば、IVS ブロードキャストは S3 バケット内の複数のオブジェクトとして保存されます。	✓		
SDK	ソフトウェア開発キットは、IVS を使用してアプリケーションを構築する開発者向けのライブラリ集です。	✓	✓	✓
自撮りセグメンテーション	カメラ画像を入力として受け取り、画像の各ピクセルがフォアグラウンドかバックグラウンドかを示す信頼度スコアを提供するマスクを返す、お客様固有のソリューションを使用して、ライブストリームのバックグラウンドを置き換えることができます 「IVS Broadcast SDK: サードパーティーのカメラフィルター (リアルタイムストリーミング)」の「背景の置換」を参照してください。		✓	
セマンティックバージョニング	Major.Minor.Patch 形式のバージョンフォーマット。API に影響しないバグ修正ではパッチバージョンが上がり、下位互換性のある API を追加または変更するとマイナーバージョンが上がり、下位互換性のない API の変更ではメジャーバージョンが上がります。	✓	✓	✓

言葉	説明	LL	RT	チャット
サーバーサイドコンポジション	IVS サーバーを使用してステージ参加者全員からの音声と動画を合成し、IVS チャネル に送信し、より多くの視聴者や S3 バケットに保存します。サーバーサイドコンポジションにより、クライアントの負荷が軽減され、ブロードキャストの耐性が向上し、帯域幅をより効率的に使用できるようになります。 「クライアント側コンポジション」も参照してください。		✓	
Service Quotas	AWS は、多くの AWS サービスのクォータを 1 か所から管理できるサービスです。クォータ値を確認できるだけでなく、Service Quotas コンソールからクォータの引き上げをリクエストすることもできます。	✓	✓	✓
サービスリンクロール	AWS サービスに直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは、IVS で自動作成され、サービスがユーザーの代わりに、S3 バケットへのアクセスなど、その他の AWS サービスを呼び出すために必要なすべてのアクセス権限が付与されます。「IVS Security」の IVS の「サービスにリンクされたロールの使用」を参照してください。	✓		
ステージ	リアルタイムのイベント参加者がリアルタイムでビデオを送受信できる仮想スペースを提示する IVS リソース。「IVS リアルタイムストリーミング入門」の「ステージの作成」を参照してください。		✓	

言葉	説明	LL	RT	チャット
ステージセッション	最初の参加者がステージに参加したときに始まり、最後の参加者がステージへの公開を停止した数分後に終了します。長期間有効なステージでは、その存続期間中に複数のセッションが発生する場合があります。		✓	
ストリーム	ソースから宛先に継続的に送信される動画またはオーディオコンテンツを表すデータ。	✓	✓	
ストリームキー	チャネル作成時に IVS によって割り当てられる識別子で、チャネルへのストリーミングの認証に使用されます。ストリームキーを使うとすべてのユーザーがチャネルにストリーミングできるため、ストリームキーは機密情報として扱ってください。「IVS 低レイテンシーストリーミングを開始する」を参照してください。	✓		
ストリームスターベーション	IVS へのストリーム配信の遅延または停止。IVS が、エンコーディングデバイスがアドバタイズした想定されたビット数を、一定期間にわたって取り込まなかった場合に発生します。ストリーム不足が発生すると、ストリームスターベーションイベントが発生します。 視聴者の観点から見ると、ストリームスタベーションは、動画における遅延、バッファリング、フリーズとして現れる場合があります。ストリームスタベーションは、ストリーム不足の原因となった特定の状況に応じて、短い (5 秒未満) 場合もあれば、長い (数分) 場合もあります。「トラブルシューティング FAQ」の「ストリームスタベーションとは」を参照してください。	✓	✓	
ストリーマー	IVS にビデオまたはオーディオストリームを送信するユーザーまたはデバイス。	✓	✓	

言葉	説明	LL	RT	チャット
サブスクライバー	ホストの動画やオーディオを受信するリアルタイムのイベント参加者。「IVS リアルタイムストリーミングとは」を参照してください。		✓	
タグ	AWS リソースに割り当てるメタデータラベル。タグを使用すると、AWS リソースの識別や整理ができます。IVS ドキュメントのランディングページでは、IVS API ドキュメント (リアルタイムストリーミング、低レイテンシーストリーミング、チャット) の「タグ付け」を参照してください。	✓	✓	✓
サードパーティーのカメラフィルター	IVS Broadcast SDK と統合できるソフトウェアコンポーネント。これにより、アプリケーションは画像をカスタム画像ソースとして Broadcast SDK に送信する前に処理できます。サードパーティのカメラフィルターは、カメラからの画像を処理したり、フィルター効果を適用したりできます。	✓	✓	
サムネイル	ストリームから取得した縮小サイズの画像。デフォルトでは、サムネイルは 60 秒ごとに生成されますが、より短い間隔を設定することもできます。サムネイルの解像度はチャネルタイプによって異なります。「Amazon S3 への自動記録 (低レイテンシーストリーミング)」の「コンテンツの記録」を参照してください。	✓		

言葉	説明	LL	RT	チャット
時間指定メタデータ	ストリーム内の特定のタイムスタンプに関連付けられたメタデータ。IVS API を使用してプログラムで追加でき、特定のフレームに関連付けられます。これにより、すべての視聴者は、ストリームに対してメタデータを同じ時点で受信できます。 時限メタデータを使用して、スポーツイベント中にチームの統計を更新するなど、クライアント側でアクションをトリガーできます。「動画ストリーム内にメタデータを埋め込む」を参照してください。	✓		
トランスコーディング	動画とオーディオを、あるフォーマットから別のフォーマットに変換します。入力ストリームが複数のビットレートと解像度が異なるフォーマットにコード変換され、各種の再生デバイスとネットワーク条件をサポートします。	✓	✓	
トランスマックス	ビデオストリームの再エンコードを行わずに、取り込んだストリームを IVS に簡単に再パッケージ化できます。トランスマックスは、「transcode multiplexing」(トランスコード多重化) の略称で、オーディオおよび/または動画ファイルを、元のストリームの一部またはすべてを保持しながら、フォーマット変更するプロセスです。トランスマキシングは、ファイルの内容を変更せずに別のコンテナ形式に変換します。トランスコーディングとは区別されます。	✓	✓	

言葉	説明	LL	RT	チャット
バリアントストリーム	同じブロードキャストを複数の異なる品質レベルでエンコードしたものです。各バリアントストリームは個別の HLS プレイリストとしてエンコードされます。利用可能なバリアントストリームのインデックスは、マルチバリアントプレイリストと呼ばれます。 IVS プレーヤーは IVS からマルチバリアントプレイリストを受信すると、再生中にバリアントストリームの中から選択でき、ネットワークの状況の変化に応じてシームレスに切り替わります。	✓		
VBR	可変ビットレート。必要なディテールのレベルに応じて再生中に変化するダイナミックビットレートを使用するエンコーダのレート制御方法です。動画品質の問題から VBR は使用しないことを強くお勧めします。代わりに CBR を使用してください。	✓	✓	

言葉	説明	LL	RT	チャット
ビュー	アクティブに動画をダウンロードまたは再生している固有の視聴セッション。視聴回数は同時視聴クォータの基準です。 視聴セッションの動画再生開始により、視聴が始まります。視聴セッションが動画の再生を停止すると、視聴が終了します。再生は視聴率の唯一の指標です。オーディオレベル、ブラウザのタブフォーカス、動画の品質などのエンゲージメントヒューリスティクスは考慮されません。IVS は視聴回数をカウントする際、個々の視聴者の正当性を考慮せず、また、1 台のマシンに複数の動画プレーヤーがあるなど、ローカライズされた視聴者の重複排除をしません。「Service Quotas (低レイテンシーストリーミング)」の「その他のクォータ」を参照してください。	✓		
表示者	IVS からストリームを受信しているユーザ。	✓		
WebRTC	ウェブリアルタイムコミュニケーションは、ウェブブラウザとモバイルアプリケーションにリアルタイムのコミュニケーションを提供するオープンソースプロジェクトです。これにより、直接通信が可能になり、オーディオとビデオ peer-to-peer の通信がウェブページ内で機能するため、プラグインをインストールしたり、ネイティブアプリケーションをダウンロードしたりする必要がありません。 WebRTC の背後にあるテクノロジーはオープンなウェブ標準として実装され、すべての主要なブラウザで通常の JavaScript APIs として、または Android や iOS などのネイティブクライアントのライブラリとして使用できます。	✓	✓	

言葉	説明	LL	RT	チャット
WHIP	WebRTC - HTTP Ingestion Protocol。ストリーミングサービスや CDN へのコンテンツの WebRTC ベースの取り込みを可能にする HTTP ベースのプロトコルです。 CDNs WHIP は、WebRTC 取り込みを標準化するために開発された IETF ドラフトです。 WHIP は OBS などのソフトウェアとの互換性を可能にし、デスクトップ公開用の代替 (IVS Broadcast SDK に) を提供します。OBS に精通している、より高度なストリーマーは、シーンの移行、オーディオミキシング、オーバーレイグラフィックスなどの高度な制作機能よりも、OBS を優先する場合があります。 WHIP は、IVS Broadcast SDK の使用が実行可能でない場合や推奨されない場合にも有益です。例えば、ハードウェアエンコーダーを含むセットアップでは、IVS Broadcast SDK はオプションではない場合があります。ただし、エンコーダーが WHIP をサポートしている場合でも、エンコーダーから IVS に直接公開できます。 「OBS と WHIP サポート」を参照してください。		✓	
WSS	WebSocket 安全なプロトコルで、暗号化された TLS 接続 WebSockets を介して を確立します。IVS Chat エンドポイントへの接続に使用されています。「IVS Chat の開始方法」の「ステップ 4: 最初のメッセージの送受信」を参照してください。			✓

ドキュメント履歴 (リアルタイムストリーミング)

リアルタイムストリーミングユーザーガイドの変更点

変更	説明	日付
OBS および WHIP サポート	新しいページを追加しました。このドキュメントでは、OBS などの WHIP 互換エンコーダーを使用して IVS リアルタイムストリーミングに発行する方法について説明します。WHIP (WebRTC - HTTP Ingestion Protocol) は、WebRTC 取り込みを標準化するために開発された IETF ドラフトです。	2024 年 2 月 6 日
Broadcast SDK: Android 1.14.1、iOS 1.14.1、Web 1.8.0	Broadcast real-time-streaming SDK ガイド: Androidhttps://docs.aws.amazon.com/ivs/latest/RealTimeUserGuide/broadcast-ios.html、iOS、および Web で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。Amazon IVS ドキュメントのランディングページでは、Broadcast SDK リファレンスのリンクが更新され、新しいバージョンを表示するようになりました。このリリースについては、「Amazon IVS リリースノート」も参照してください。	2024 年 2 月 1 日

	Android ガイドに、新しい既知の問題 (ビデオサイズが 176 x 176 未満) を追加しました。 ウェブガイドに、新しい既知の問題を追加しました。回避策は、 `getUserMedia` または を呼び出すときにビデオ解像度を 720p に制限することですgetDisplayMedia 。 「リアルタイムストリーミングの最適化」で、「サイマルキャストによるレイヤードエンコーディングの設定」を更新しました。これはデフォルトで無効になっています。	
Broadcast SDK: Android 1.13.4、iOS 1.13.4、Web 1.7.0	Broadcast real-time-streaming SDK ガイド: Androidhttps://docs.aws.amazon.com/ivs/latest/RealTimeUserGuide/broadcast-ios.html、iOS、および Web で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。Amazon IVS ドキュメントのランディングページでは、Broadcast SDK リファレンスのリンクが更新され、新しいバージョンを表示するようになりました。このリリースについては、「Amazon IVS リリースノート」も参照してください。	2024 年 1 月 3 日

IVS の用語集	用語集を拡張し、IVS のリアルタイム、低レイテンシー、チャット用語を網羅しました。	2023 年 12 月 20 日
ステージヘルス: 新しい CloudWatch メトリクス	PacketLoss (Stage) メトリクスの名前を DownloadPacketLoss (Stage) に変更し、IVS リアルタイムストリーミング用の追加の CloudWatch メトリクスをリリースしました。 • DownloadPacketLoss（ステージ、参加者） • DroppedFrames（ステージ、参加者） • SubscribeBitrate（ステージ、参加者、MediaType） 「IVS リアルタイムストリーミングのモニタリング」を参照してください。	2023 年 12 月 7 日

IAM マネージドポリシー	IVS ReadOnlyAccess と IVS の 2 つのマネージドポリシーを追加しましたFullAccess。以下を参照してください。 • セキュリティページの Amazon IVS 用マネージドポリシーに関する新しいセクション。 • 「IVS 低レイテンシーストリーミングを開始する」の「ステップ 3: IAM アクセス権限の設定」への変更。	2023 年 12 月 5 日
Broadcast SDK: Android 1.13.2、iOS 1.13.2	Broadcast real-time-streaming SDK ガイド: Android および iOS で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。 Amazon IVS ドキュメントのランディングページでは、Broadcast SDK リファレンスのリンクが更新され、新しいバージョンを表示するようになりました。 このリリースについては、「Amazon IVS リリースノート」も参照してください。	2023 年 12 月 4 日

Broadcast SDK: Android 1.13.1	Broadcast real-time-streaming SDK ガイド: Android で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。 Amazon IVS ドキュメントのランディングページでは、Broadcast SDK リファレンスのリンクが更新され、新しいバージョンを表示するようになりました。 このリリースについては、「Amazon IVS リリースノート」も参照してください。	2023 年 11 月 21 日
Service Quotas	「参加者が公開する解像度」を 1080p から 720p に変更しました。	2023 年 11 月 18 日

Broadcast SDK: Android 1.13.0、iOS 1.13.0	Broadcast real-time-streaming SDK ガイド: Android および iOS で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。 Amazon IVS ドキュメントのランディングページでは、Broadcast SDK リファレンスのリンクが更新され、新しいバージョンを表示するようになりました。 このリリースについては、「Amazon IVS リリースノート」も参照してください。 また、ストリーミングの最適化にもさまざまな更新を行いました。とりわけ、「アダプティブストリーミング: サイマルキャストによるレイヤードエンコーディング」機能には明示的なオプトインが必要になり、最近のバージョンの SDK でのみサポートされるようになりました。	2023 年 11 月 17 日

Composite Recording	以下の変更を加えました。 • この新機能に関する「Composite Recording」ページを追加しました。 • S3 エンドポイントでの「IVS リアルタイムストリーミングの開始」では、「IAM アクセス許可を設定する」のポリシーを更新しました。 • 「Service Quotas」で、新しいエンドポイントのコールレートクォータを更新しました。	2023 年 11 月 16 日

サーバーサイドコンポジション (SSC)	IVS サーバーサイドコンポジションにより、クライアントは IVS ステージのコンポジションとブロードキャストを IVS が管理するサービスにオフロードできます。チャネルへの SSC および RTMP ブロードキャストは、ステージのホームリージョンにある IVS コントロールプレーンエンドポイントを介して呼び出されます。以下を参照してください。 • 開始 – 「IAM アクセス許可を設定する」のポリシーに SSC エンドポイントを追加しました。 • IVS での Amazon EventBridge の使用 — 新しいメトリクスを追加しました。 • サーバーサイドコンポジション – この新しいドキュメントに概要とセットアップ手順を記載しました。 • Service Quotas – 新しいコールレート制限とその他のクォータを追加しました。 以下も参照してください。 • 以下の「IVS リアルタイムストリーミング API リファレンスの変更」にリストアップされている変更。	2023 年 11 月 16 日

	• 「ドキュメント履歴 (低レイテンシーストリーミング)」にリストアップされている変更。	
IVS Broadcast SDK	Broadcast SDK の概要では、サポートされている SDK バージョンを明確にするために「プラットフォーム要件」の「ネイティブプラットフォーム」を更新し、「モバイルブラウザ (iOS と Android)」を追加しました。 ブロードキャストウェブガイドに「モバイルウェブの制限事項」を追加しました。	2023 年 11 月 9 日
IVS Broadcast SDK	サードパーティー製カメラフィルターに関する新しいページを追加しました。	2023 年 11 月 9 日
IVS リアルタイムストリーミングの開始	「IAM アクセス許可を設定する」の手順が更新されました。	2023 年 10 月 20 日
リアルタイムストリーミングのモニタリング	CloudWatch メトリクス: IVS リアルタイムストリーミングで、ディメンションのサンプル値を追加しました。	2023 年 10 月 17 日
Broadcast SDK: Web ガイド	「リモート参加者のメディアミュート状態の監視」にいくつかの変更が加えられました。	2023 年 10 月 17 日

Broadcast SDK: Web 1.6.0	Broadcast real-time-streaming SDK ガイド: Web で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。 Amazon IVS ドキュメントのランディングページは、ブロードキャスト SDK リファレンスの最新バージョンを参照しています。 このリリースについては、「Amazon IVS リリースノート」も参照してください。 ウェブガイドの MediaStream「デバイスから を取得する」で、2 つのmax行も削除されました。ベストプラクティスは のみを指定することですideal。 「リアルタイムストリーミングの最適化」に「オーディオビットレートとステレオサポートの最適化」という新しいセクションを追加しました。	2023 年 10 月 16 日
ステージヘルス: 新しい CloudWatch メトリクス	IVS リアルタイムストリーミングの CloudWatch メトリクスをリリースしました。「IVS リアルタイムストリーミングのモニタリング」を参照してください。	2023 年 10 月 12 日

Broadcast SDK: Android 1.12.1	Broadcast real-time-streaming SDK ガイド: Android で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。また、「Bluetooth マイクの使用」という新しいセクションも追加されました。 Amazon IVS ドキュメントのランディングページは、ブロードキャスト SDK リファレンスの最新バージョンを参照しています。 このリリースについては、「Amazon IVS リリースノート」も参照してください。	2023 年 10 月 12 日
Broadcast SDK: Web 1.5.2	Broadcast real-time-streaming SDK ガイド: Web で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。 Amazon IVS ドキュメントのランディングページは、ブロードキャスト SDK リファレンスの最新バージョンを参照しています。 このリリースについては、「Amazon IVS リリースノート」も参照してください。	2023 年 9 月 14 日

IVS リアルタイムストリーミングの開始	「Android」>「Broadcast SDK のインストール」にデータバインディングを追加しました。	2023 年 9 月 12 日
Broadcast SDK エラー処理	ブロードキャスト SDK ガイド (Web、Android、および iOS) に「エラー処理」セクションを追加しました。	2023 年 9 月 12 日
IVS リアルタイムストリーミングの開始	参加者トークンの配布で、現在のトークン形式に基づいて機能を構築しないことに関する重要な注意事項が追加されました。	2023 年 9 月 1 日
IVS リアルタイムストリーミングの開始	IAM アクセス許可の設定で、権限のセットを更新しました。	2023 年 8 月 31 日

Broadcast SDK: Web 1.5.1、Android 1.12.0、iOS 1.12.0	Broadcast real-time-streaming SDK ガイド: Web https://docs.aws.amazon.com/ivs/latest/RealTimeUserGuide/broadcast-android.html、Android、iOS で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。 Amazon IVS ドキュメントのランディングページでは、Broadcast SDK リファレンスのリンクが更新され、新しいバージョンを表示するようになりました。 このリリースについては、「Amazon IVS リリースノート」も参照してください。	2023 年 8 月 23 日

リアルタイムストリーミングのローンチ	このリリースに伴い、ドキュメントに大幅な変更が行われました。以前のドキュメントの名前を「IVS 低レイテンシーストリーミング」に変更し、新しい IVS リアルタイムストリーミングドキュメントを公開しました。「IVS ドキュメントのランディングページ」に、リアルタイムストリーミングと低レイテンシーストリーミングの個別のセクションが追加されました。各セクションには、それぞれのユーザーガイドと API リファレンスがあります。 その他のドキュメントの変更については、「ドキュメント履歴 (低レイテンシーストリーミング)」を参照してください。	2023 年 8 月 7 日

Broadcast SDK: Web 1.5.0、Android 1.11.0、iOS 1.11.0	ブロードキャスト SDK ガイド: Web、Android、および iOS で、新しいリリースのバージョン番号とアーティファクトのリンクが更新されました。 Amazon IVS ドキュメントのランディングページでは、Broadcast SDK リファレンスのリンクが更新され、新しいバージョンを表示するようになりました。 このリリースについては、「Amazon IVS リリースノート」も参照してください。	2023 年 8 月 7 日

IVS リアルタイムストリーミング API リファレンスの変更

API の変更	説明	日付
Composite Recording	4 つの StorageConfiguration エンドポイントと 7 つのオブジェクト (DestinationDetail、、S3DestinationConfiguration RecordingConfiguration、SS3Detail, S3StorageConfiguration StorageConfiguration、) を追加しました StorageConfigurationSummary。 3 つのオブジェクト (コンポジション、送信先、) を変更しました DestinationConfiguration。これは、GetComposition レスポンスと StartComposition リクエストとレスポンスに影響します。	2023 年 11 月 16 日

API の変更	説明	日付
サーバーサイドコンポジション	8 つのコンポジションと EncoderConfiguration エンドポイント、および 11 のオブジェクト (ChannelDestinationConfiguration、コンポジション、送信先 CompositionSummary、 DestinationConfiguration、 DestinationSummary EncoderConfiguration、EncoderConfigurationSummary、 GridConfiguration LayoutConfiguration、ビデオ) を追加しました。	2023 年 11 月 16 日
ステージヘルス: 新規参加者データ	参加者オブジェクトに 6 つのフィールド (`browserName` 、`browserVersion` 、`ispName`、`osName`、`osVersion` 、`sdkVersion`) が追加されました。これは GetParticipant レスポンスに影響します。	2023 年 10 月 12 日
参加者トークン	現在のトークン形式に基づいて機能を構築しないことに関する重要な注意事項が追加されました。	2023 年 9 月 1 日
IVS リアルタイムストリーミングのローンチ	このリリースに伴い、ドキュメントに大幅な変更が行われました。以前のドキュメントの名前を「IVS 低レイテンシーストリーミング」に変更し、新しい IVS リアルタイムストリーミングドキュメントを公開しました。「IVS ドキュメントのランディングページ」に、リアルタイムストリーミングと低レイテンシーストリーミングの個別のセクションが追加されました。各セクションには、それぞれのユーザーガイドと API リファレンスがあります。 「IVS リアルタイムストリーミング API リファレンス」は IVS リアルタイムストリーミングドキュメントの一部です。以前のタイトルは「IVS ステージ API リファレンス」でした。これまでの履歴については、「ドキュメント履歴 (低レイテンシーストリーミング)」に記載されています。	2023 年 8 月 7 日

リリースノート (リアルタイムストリーミング)

2024 年 2 月 6 日

OBS および WHIP サポート

IVS は、OBS などの WHIP 互換エンコーダーと併用して、IVS リアルタイムストリーミングに公開できます。WHIP (WebRTC - HTTP Ingestion Protocol) は、WebRTC 取り込みを標準化するために開発された IETF ドラフトです。OBS と WHIP サポートに関する新しいページを参照してください。

2024 年 2 月 1 日

Amazon IVS Broadcast SDK: Android 1.14.1、iOS 1.14.1、Web 1.8.0 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
Web Broadcast SDK 1.8.0	リファレンスドキュメント： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • サイマルキャストによるレイヤードエンコーディングがデフォルトで無効になりました。 • ステージが削除されたとき、または参加者がサーバーから切断されたときに、ステージインスタンスがクリーンに切断されない問題を修正しました。SDK は、`DISCONNECTED` (および ではなく) の状態の`STAGE_CONNECTION_STATE_CHANGED` イベントを発行`ERRORED`するようになりました`CONNECTING` 。 • 空のオーディオトラックまたはビデオトラックで戦略を更新すると、公開が失敗する問題を修正しました。

プラットフォーム	ダウンロードおよび変更
Android Broadcast SDK 1.14.1	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/android • サイマルキャストによるレイヤードエンコーディングがデフォルトで無効になりました。 • M108 libWebRTC から M119 に更新されました。 • 全体的な安定性を向上させるために、いくつかのクラッシュを修正しました。 • ステレオパブリッシングのサポートを追加しました。これは、 StageAudioConfiguration オブジェクトを介して有効にできます。 • セッションに参加した後、参加者からのブラックフィードが発生するバグを修正しました。 • 同じホストアプリケーションに他のlibWebRTC バージョンが含まれている場合の記号の競合を避けるため、内部libWebRTC リファレンスを更新しました。

プラットフォーム	ダウンロードおよび変更
iOS Broadcast SDK 1.14.1	リアルタイムストリーミングのダウンロード: https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/ios • サイマルキャストによるレイヤードエンコーディングがデフォルトで無効になりました。 • M108 libWebRTC から M119 に更新されました。 • 全体的な安定性を向上させるために、いくつかのクラッシュを修正しました。 • ステレオパブリッシングのサポートを追加しました。これは、 を通じて有効にできますIVSLocalStageStreamAudioConfiguration 。 • 他の参加者のオーディオ専用モードを有効にするとクラッシュする問題を修正しました。 • TTV を改善し、バイナリサイズを縮小しました。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

Broadcast SDK サイズ: iOS

アーキテクチャ	圧縮サイズ	非圧縮サイズ
arm64	3.350 MB	7.790 MB

2024 年 1 月 3 日

Amazon IVS Broadcast SDK: Android 1.13.4、iOS 1.13.4、Web 1.7.0 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
Web Broadcast SDK 1.7.0	リファレンスドキュメント： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • ステージに参加するサブスクライバー time-to-video 向けの が改善されました。 • `minAudioBitrateKbps` プロパティを削除しました (使用されていませんでした)。 • インターネットの停止または変更中のネットワーク復旧が改善されました。
Android Broadcast SDK 1.13.4	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/android • StageAudioConfiguration では、エコーキャンセルを有効にするかどうかの設定がサポートされるようになりました。
iOS Broadcast SDK 1.13.4	リアルタイムストリーミング用のダウンロード: https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip

プラットフォーム	ダウンロードおよび変更
	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/ios • iOS では、安定性と回復可能性に焦点を当てて、録音と再生の両方のオーディオエンジンを改善しました。これにより、使用中のルート変更のサポートが向上し、エッジケースのバッテリー復旧が改善され、メインスレッドブロックの量が減ります。 • ステージからデタッチした後もマイクがアクティブのままになり、iOS プライバシーインジケータがオンのままになることがある問題を修正しました。(SDK は、その時点で受信音声を処理していませんでした)。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

Broadcast SDK サイズ: iOS

アーキテクチャ	圧縮サイズ	非圧縮サイズ
arm64	3.45 MB	7.84 MB

2023 年 12 月 7 日

新しい CloudWatch メトリクス

PacketLoss (Stage) メトリクスの名前を DownloadPacketLoss (Stage) に変更しました。また、IVS リアルタイムストリーミングに関する追加の CloudWatch メトリクスもリリースされました。

- DownloadPacketLoss（ステージ、参加者）
- DroppedFrames（ステージ、参加者）
- SubscribeBitrate（ステージ、参加者、MediaType）

詳細については、「IVS リアルタイムストリーミングのモニタリング」を参照してください。

2023 年 12 月 4 日

Amazon IVS Broadcast SDK: Android 1.13.2、iOS 1.13.2 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
すべてのモバイル (Android と iOS)	• 開発者はノイズ抑制設定を使用してパブリッシングを有効/無効にできます。
Android Broadcast SDK 1.13.2	リファレンスドキュメント：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/android • セッションの最初のステージに参加するときの動画 (TTV) の読み込みにかかる時間が改善されました。
iOS Broadcast SDK 1.13.2	リアルタイムストリーミングのダウンロード: https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/ios

プラットフォーム	ダウンロードおよび変更
	• リアルタイム SDK には変更はありません。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

Broadcast SDK サイズ: iOS

アーキテクチャ	圧縮サイズ	非圧縮サイズ
arm64	3.45 MB	7.82 MB

2023 年 11 月 21 日

Amazon IVS Broadcast SDK: Android 1.13.1 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
Android Broadcast SDK 1.13.1	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.13.1/android • 同じステージに対して急速に退出、リリース、再参加するとクラッシュする問題を修正しました。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

2023 年 11 月 17 日

Amazon IVS Broadcast SDK: Android 1.13.0、iOS 1.13.0 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
すべてのモバイル (Android と iOS)	• 「ストリーミングの最適化」を更新しました。とりわけ、「アダプティブストリーミング: サイマルキャストによるレイヤードエンコーディング」機能には明示的なオプトインが必要になり、最近のバージョンの SDK でのみサポートされるようになりました。 • まれに発生するクラッシュを削減することで、ステージの安定性を向上させました。 • ステージに参加するときの動画 (TTV) の読み込みにかかる時間が改善されました。 • Bluetooth デバイスでの操作性が向上しました。 • SDK の CPU とメモリの使用量を最適化し、ライブラリのサイズを縮小しました。 • `StageAudioManager` クラスを追加しました。このクラスを使用して、音声コミュニ

プラットフォーム	ダウンロードおよび変更
	ケーションやメディア再生などのプリセットを含む、オーディオのキャプチャと再生のパラメータを設定できます。詳細については、新しいページ「IVS Broadcast SDK: Mobile Audio Modes」を参照してください。 • WebRTC 統計から構造化された品質イベントを表示する新しい `requestQualityStats` 関数を追加しました。 • オーディオビットレートを更新する新しい機能を追加しました。動画の設定と同じように `LocalStageStream` オブジェクトに設定されますが、新しいオーディオ設定オブジェクトによって設定されます。

プラットフォーム	ダウンロードおよび変更

Android Broadcast SDK 1.13.0	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/android
	• StageRenderer インターフェイス上のメソッドがすべてオプションになりました。
	• パフォーマンスを向上させるため、Surfaceview ベースのプレビューのサポートが追加されました。Session および StageStream の既存の getPreview メソッドは引き続き TextureView のサブクラスを返しますが、これは今後の SDK バージョンで変更される可能性があります。
	  • アプリケーションが特に TextureView に依存している場合は、変更せずに続行できます。getPreview から getPreviewTextureView に切り替えて、デフォルトの getPreview が返す内容がいつかは変更されることに備えることもできます。
	  • アプリケーションが特に TextureView を必要としない場合は、CPU とメモリの使用量を抑えるために getPreviewSurfaceView に切り替えることをお勧めします。
	• アプリケーション提供の Android Surface オブジェクトで動作する ImagePreviewSurfaceTarget という新しいタイプのプレビューが SDK に実装されました。これは Android View のサブクラスではないため、柔軟性が向上しています。
	• リモート参加者の onFrame コールバックが間違った時間に間違ったサイズで呼び出されるケースを修正しました。

プラットフォーム	ダウンロードおよび変更
	• SurfaceSource # getInputSurface に @Nullable という注釈が付くようになりました。使用する前にコードで確認する必要があります。 • UserId および attributes が ParticipantInfo に追加されました。UserId プロパティと attributes プロパティはトークンに埋め込まれており、参加者が参加するたびにアプリケーションは ParticipantInfo を通じてそれらのプロパティを取得できます。 • カメラのキャプチャとプレビューのレンダリングが、デフォルトで 720 x 1280 または公開の解像度 (どちらか大きい方) で 15 fps になりました。解像度や fps は StageVideoConfiguration # setCameraCaptureQuality を使用して調整できます。 • 設定プロパティを設定する際にスローされる IllegalArgumentException に、例外メッセージで指定された値が含まれるようになりました。

プラットフォーム	ダウンロードおよび変更
iOS Broadcast SDK 1.13.0	リアルタイムストリーミングのダウンロード: https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/ios • 公開前に動画の設定を更新しても SDK によって動画の設定が変更されない問題を修正しました。 • LibVPX のセキュリティ脆弱性 (CVE-2023-5217) に対する Google の修正が組み込まれました。(Android SDK ではこの問題に対処するための変更は必要なかったことに注意してください) • `libWebRTC` を含む他のライブラリを使用するアプリケーションが IVS Broadcast SDK と競合しなくなりました。 • これで、`IVSStageRenderer` プロトコル上のすべてのメソッドが `@optional` とマークされます。 • SDK 自体に記載されているように、SDK から返されたマイクとカメラでソート順序が保証されるようになりました。 • 複数のカメラの `isDefault` プロパティに `true` の値を設定できるようになりました。これは、オペレーティングシステムによって決定される位置ごとに 1 つになります。 • `IVSStageAudioManager` が追加されたことで、基盤となる `AVAudioSession` を正確に制御できるようになり、ステージ機能のさまざまなユースケースに対応できるようになりました。

プラットフォーム	ダウンロードおよび変更
	• `UserId`が`ParticipantInfo` に追加されました。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

Broadcast SDK サイズ: iOS

アーキテクチャ	圧縮サイズ	非圧縮サイズ
arm64	3.45 MB	7.84 MB

2023 年 11 月 16 日

Composite Recording

この新機能により、IVS ステージの複合ビューが Amazon S3 バケットに録画されます。詳細については、以下を参照してください。

- Composite Recording – これは新しいページです。
- IVS リアルタイムストリーミングの開始 – 「IAM アクセス許可を設定する」でポリシーに S3 エンドポイントを追加しました。
- Service Quotas – 新しいエンドポイントにコールレートクォータを追加しました。

- IVS リアルタイムストリーミング API リファレンス — 4 つの StorageConfiguration エンドポイントと 7 つのオブジェクト (DestinationDetail、S3DestinationConfiguration RecordingConfiguration、S3Detail, S3StorageConfiguration、、) を追加しました StorageConfiguration StorageConfigurationSummary。また、3 つのオブジェクト (コンポジション、送信先 DestinationConfiguration) も変更しました。これは GetComposition レスポンスと StartComposition リクエストとレスポンスに影響します。

2023 年 11 月 16 日

サーバーサイドコンポジション

IVS サーバーサイドコンポジションにより、クライアントは IVS ステージのコンポジションとブロードキャストを IVS が管理するサービスにオフロードできます。サーバーサイドコンポジションとチャネルへの RTMP ブロードキャストは、ステージのホームリージョンにある IVS コントロールプレーンエンドポイントを介して呼び出されます。詳細については、以下を参照してください。

- IVS リアルタイムストリーミングの開始 – 「IAM アクセス許可を設定する」でポリシーに SSC エンドポイントを追加しました。
- IVS リアルタイムストリーミング EventBridge での Amazon の使用 — 新しいメトリクスを追加しました。
- サーバーサイドコンポジション – この新しいドキュメントに概要とセットアップ手順を記載しました。
- Service Quotas (リアルタイムストリーミング) – 新しいコールレート制限とその他のクォータを追加しました。
- リアルタイムストリーミング API リファレンス – 8 つのコンポジションと EncoderConfiguration エンドポイント、および 11 のオブジェクト (ChannelDestinationConfiguration、コンポジション、 CompositionSummary、宛先 DestinationConfiguration、 DestinationSummary、EncoderConfiguration、 LayoutConfiguration、 GridConfiguration、ビデオ) EncoderConfigurationSummaryを追加しました。

「Amazon IVS 低レイテンシーストリーミングユーザーガイド」で以下を参照して下さい。

- IVS ストリームで複数ホストを有効にする – 「ステージのブロードキャスト: クライアントサイドコンポジションとサーバーサイドコンポジション」を追加し、「4. ステージをブロードキャストする」を更新しました。

2023 年 10 月 16 日

Amazon IVS Broadcast SDK: Web 1.6.0 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
Web Broadcast SDK 1.6.0	リファレンスドキュメント： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • Time-To-Video (TTV) が改善されました。 • 最大 128kbps のモノラルまたはステレオオーディオチャンネルをサポートする `maxAudioBitrate` が追加されました。

2023 年 10 月 12 日

新しい CloudWatch メトリクスと参加者データ

IVS リアルタイムストリーミングの CloudWatch メトリクスをリリースしました。詳細については、「IVS リアルタイムストリーミングのモニタリング」を参照してください。

参加者 API オブジェクトに 6 つのフィールド (`browserName`、`browserVersion`、`ispName`、`osName`、`osVersion`、`sdkVersion`) が追加されました。これは GetParticipant レスポンスに影響します。「IVS リアルタイムストリーミング API リファレンス」を参照してください。

2023 年 10 月 12 日

Amazon IVS Broadcast SDK: Android 1.12.1 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
Android Broadcast SDK 1.12.1	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.12.1/android

プラットフォーム	ダウンロードおよび変更
	• `BroadcastSession.setListener` の呼び出しがエラーになるバグが修正されました。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

2023 年 9 月 14 日

Amazon IVS Broadcast SDK: Web 1.5.2 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
Web Broadcast SDK 1.5.2	リファレンスドキュメント： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 公開状態が `ERRORED` 状態になった場合に `refreshStrategy` で再公開できないバグを修正しました。

2023 年 8 月 23 日

Amazon IVS Broadcast SDK: Web 1.5.1、Android 1.12.0、iOS 1.12.0 (リアルタイムストリーミング)

プラットフォーム	ダウンロードおよび変更
Web Broadcast SDK 1.5.1	リファレンスドキュメント： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 内部 Maybe 型が 5 のバグを修正しました TypeScript。 • サイマルキャストサポートの検出機能が向上しました。 • 公開時に発生する `refreshStrategy` による 2 つの競合状態を修正しました。 • 登録する参加者の更新時に発生する `refreshStrategy` との競合状態を修正しました。
すべてのモバイル (Android と iOS)	• 公開アクションが完了しないというまれな問題を修正しました。 • まれに発生するクラッシュを削減することで、ステージの安定性を向上させました。 • 急速な参加/脱退による競合状態の問題を解決し、ステージの安定性が向上させました。 • 新しい `setOnFrameCallback` メソッドを `ImageDevice` に追加しました。これにより、フレームがデバイス自体を通過する様子を観察できるようになり、最新の画像のアスペクト比を把握できます。このメソッドを使用すると、ステージ内のリモート参加者の最初のフレームがいつレンダリングされるかを検出することもできます。

プラットフォーム	ダウンロードおよび変更
Android ブロードキャスト SDK 1.12.0	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/android • Android 9 がサポートされるようになりました。 • CPU 使用率とパフォーマンスが向上しました。
iOS Broadcast SDK 1.12.0	リアルタイムストリーミングのダウンロード: https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/ios • `IVSDeviceDiscovery.createAudioSourceWithName` の署名を `IVSCustomImageSource` ではなく、`IVSCustomAudioSource` を返すように修正しました。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

Broadcast SDK サイズ: iOS

アーキテクチャ	圧縮サイズ	非圧縮サイズ
arm64	5.06 MB	10.92 MB

2023 年 8 月 7 日

Amazon IVS Broadcast SDK: Web 1.5.0、Android 1.11.0、iOS 1.11.0

プラットフォーム	ダウンロードおよび変更
Web Broadcast SDK 1.5.0	リファレンスドキュメント： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • サイマルキャストの追加 — この機能を有効にすると、パブリッシャーは高画質および低画質の動画レイヤーを送信できます。サブスクライバーは、ネットワークの状態に基づいて最適な品質を自動的に選択します。「メディアの最適化」を参照してください。
すべてのモバイル (Android と iOS)	サイマルキャストの追加 — この機能を有効にすると、パブリッシャーは高画質および低画質の動画レイヤーを送信できます。サブスクライバーは、ネットワークの状態に基づいて最適な品質を自動的に選択します。Android および iOS ブロードキャスト SDK ガイドの「サイマルキャストでのレイヤードエンコーディングの有効化/無効化」を参照してください。
Android Broadcast SDK 1.11.0	リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/android

プラットフォーム	ダウンロードおよび変更
	• ステージを多数作成すると最終的にクラッシュする問題を修正しました。(正確なステージ数はデバイスによって異なります。)
iOS Broadcast SDK 1.11.0	リアルタイムストリーミング用のダウンロード: https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント： https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/ios • `IVSDeviceDiscovery.createAudioSourceWithName` の署名を `IVSCustomImageSource` ではなく、`IVSCustomAudioSource` を返すように修正しました。

Broadcast SDK サイズ: Android

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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

Broadcast SDK サイズ: iOS

アーキテクチャ	圧縮サイズ	非圧縮サイズ
arm64	5.030 MB	10.810 MB

2023 年 8 月 7 日

リアルタイムストリーミング

Amazon Interactive Video Service (IVS) リアルタイムストリーミングを使用すると、ホストから視聴者まで 300 ミリ秒未満のレイテンシーでライブストリームを配信できます。

このリリースに伴い、ドキュメントに大幅な変更が行われました。IVS ドキュメントのランディングページに、リアルタイムストリーミングと低レイテンシーストリーミングの個別のセクションが追加されました。各セクションには、それぞれのユーザーガイドと API リファレンスがあります。ドキュメントの詳細については、ドキュメント履歴を参照してください (リアルタイムと低レイテンシーのドキュメントの変更)。リアルタイムストリーミングの場合は、最初に「IVS リアルタイムストリーミングユーザーガイド」および「IVS リアルタイムストリーミング API リファレンス」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。