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 Real-Time Streaming 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 Real-Time Streaming 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 Real-Time Streaming 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。25,000 人以上の視聴者に対応、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。(デフォルトの制限や、引き上げをリクエストする手順については、Service Quotas でリアルタイムストリーミングおよび低レイテンシーストリーミングを参照してください)
- モバイル最適化 — IVS Real-Time Streaming 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 Real-Time Streaming 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

IVS Real-Time Streaming の概要

このセクションでは、Real-Time Streaming を使用するための前提条件の一覧を示し、主要な用語を紹介します。

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 Real-Time Streaming 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. 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許可を付与する 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 (IVS console、the section called “Web”、the section called “Android”、the section called “iOS”)。

ステップ 1: 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 のセキュリティ」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 のセキュリティ」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、アプリケーションデベロッパーにすべての IVS Get および List API オペレーション (低レイテンシーおよびリアルタイムストリーミングの両方) へのアクセスを許可します。
- `IVSFullAccess` AWS マネージドポリシーを使用して、アプリケーションデベロッパーにすべての IVS API オペレーション (低レイテンシーおよびリアルタイムストリーミングの両方) へのアクセスを許可します。

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

以下の手順に従ってください。

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 アクションは含まれていません。必要に応じてオペレーションアクセス許可を追加/削除 (許可/拒否) できます。IVS オペレーションの詳細については、「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)

   JSON

```
{
   "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

以下の手順に従ってください。

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] を選択します。[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

以下の手順に従ってください。

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。

ステップ 2: ステージを作成する

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 オペレーションのいずれかを使用し、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。

Amazon IVS コンソールまたは AWS CLI を介して、個々の参加者による記録の有無を問わずにステージを作成できます。ステージの作成および記録については、以下で説明します。

個々の参加者の録画

ステージの個々の参加者による記録を有効にするオプションがあります。個々の参加者の記録における S3 機能が有効になっている場合、ステージへの個々の参加者によるブロードキャストはすべて記録され、ユーザーが所有する Amazon S3 ストレージバケットに保存されます。その後、この録画はオンデマンド再生が可能です。

このセットアップは拡張オプションです。デフォルトでは、ステージの作成時に記録は無効になっています。

記録用にステージを設定する前に、保存設定を作成する必要があります。ステージの記録されたストリームが保存される Amazon S3 ロケーションを指定するリソースです。コンソールまたは CLI を使用して保存設定を作成および管理することができます。両方の手順を以下に示されます。保存設定

を作成したら、ステージ作成時 (以下で説明) または既存のステージを更新することで後で作成する際、保存設定をステージに関連付けします。(API で「CreateStage」および「UpdateStage」を参照してください) 複数のステージを同じ保存設定に関連付けることができます。すべてのステージに関連付けされなくなった保存設定を削除できます。

次の制約に注意が必要です。

- S3 バケットを所有している必要があります。つまり、記録されるステージを設定するアカウントは、記録を保存する S3 バケットを所有している必要があります。
- ステージ、保存設定、S3 ロケーションは同じ AWS リージョンにある必要があります。他のリージョンでステージを作成して記録する場合、それらのリージョンで保存設定および S3 バケットも設定する必要があります。

S3 バケットに録画するには、AWS 認証情報を使用した認可が必要です。IVS に必要なアクセス権を付与するため、録画設定が作成される際に AWS IAM サービスにリンクされたロール (SLR) が自動的に作成されます。SLR は、特定のバケットのみに IVS 書き込み許可を付与するように制限されます。

ストリーミングロケーションおよび AWS 間、あるいは AWS 内のネットワーク問題により、ストリームの記録中に一部データが失われる可能性があることに注意してください。このような場合、Amazon IVS は録画よりもライブストリームを優先します。冗長性のために、ストリーミングツールでローカルに録画してください。

記録したファイルのポスト処理や VOD 再生の設定方法などの詳細については「個々の参加者の録画」を参照してください。

録画を無効にする方法

既存のステージで Amazon S3 記録を無効にする方法

- コンソール — 関連するステージの詳細ページの個々の参加者のストリーミングを記録するセクションで、[S3 への自動録画] の [自動録画の有効化] をオフに切り替え、[変更を保存する] を選択します。保存設定によるステージとの関連付けが解除され、ステージ上のストリームが記録されなくなります。
- CLI — `update-stage`コマンドを実行し、録画設定の ARN を空の文字列として渡します。

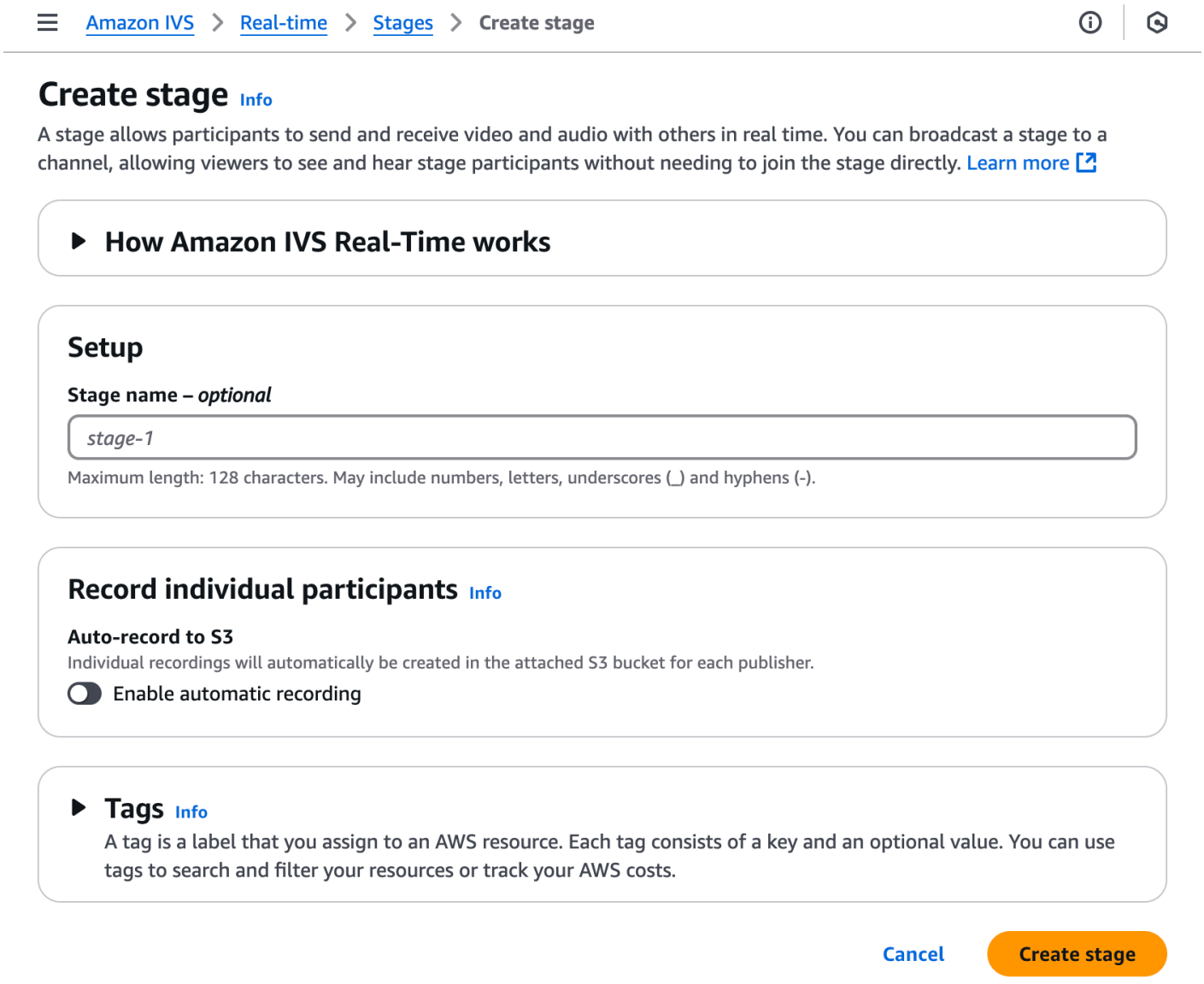
```
aws ivs-realtime update-stage --arn arn:aws:ivs:us-west-2:123456789012:stage/
abcdABCDefgh --auto-participant-recording-configuration storageConfigurationArn=""
```

記録が無効になったことを示す `storageConfigurationArn` の空の文字列が含まれるステージオブジェクトが返されます。

IVS ステージを作成するためのコンソール手順

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。



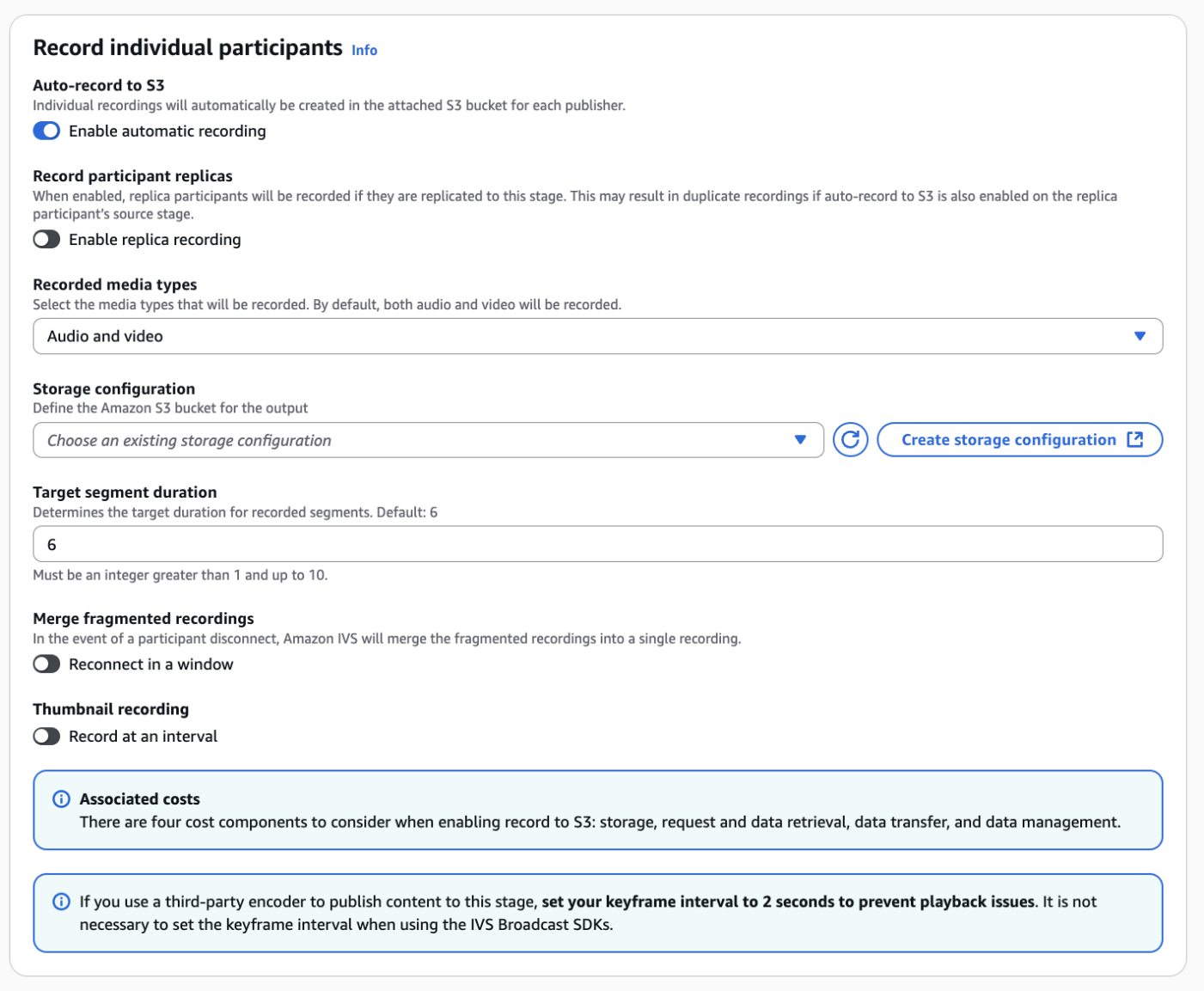
3. 必要に応じて、[ステージ名] を入力します。

4. 個々の参加者の記録を有効にする場合、以下の「Amazon S3 に個々の参加者の自動記録を設定する (オプション)」の手順を完了させてください。
5. [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。

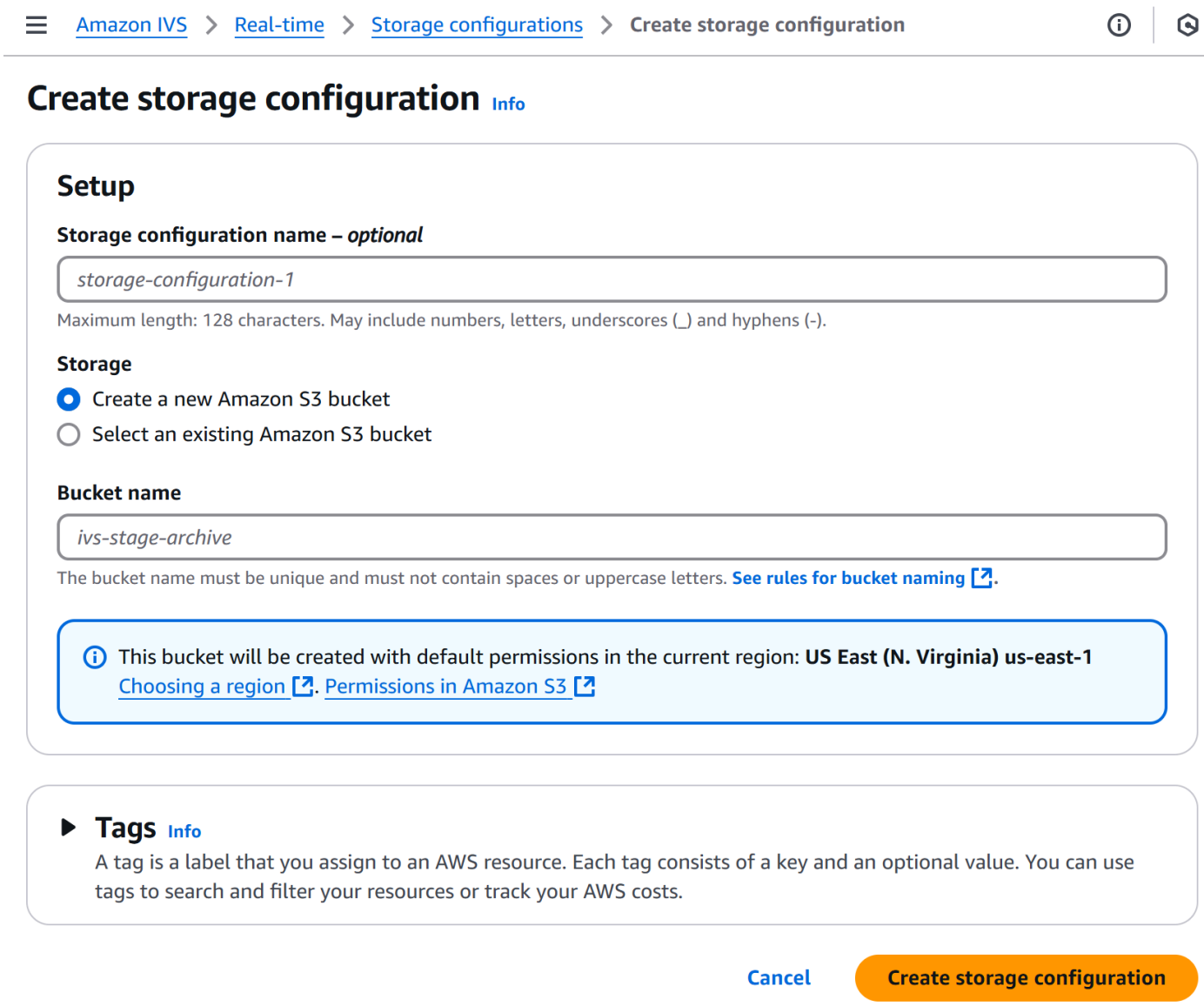
Amazon S3 に個々の参加者の自動記録を設定する (オプション)

ステージの作成中に個々の参加者の記録を有効にするには、次の手順に従ってください。

1. ステージを作成するページの個々の参加者を記録するで、[自動録画の有効化] をオンにします。[記録済みメディアタイプ] を選択し、既存の [保存設定] を選択するか新しい保存設定を作成し、ある時間間隔でサムネイルを記録するかどうかを選択するため、追加のフィールドが表示されます。



2. 記録するメディアタイプを選択します。
3. [保存設定を作成する] を選択します。新しいウィンドウが開きます。オプションを使用して、Amazon S3 バケットを作成し、新しい録画設定にアタッチします。

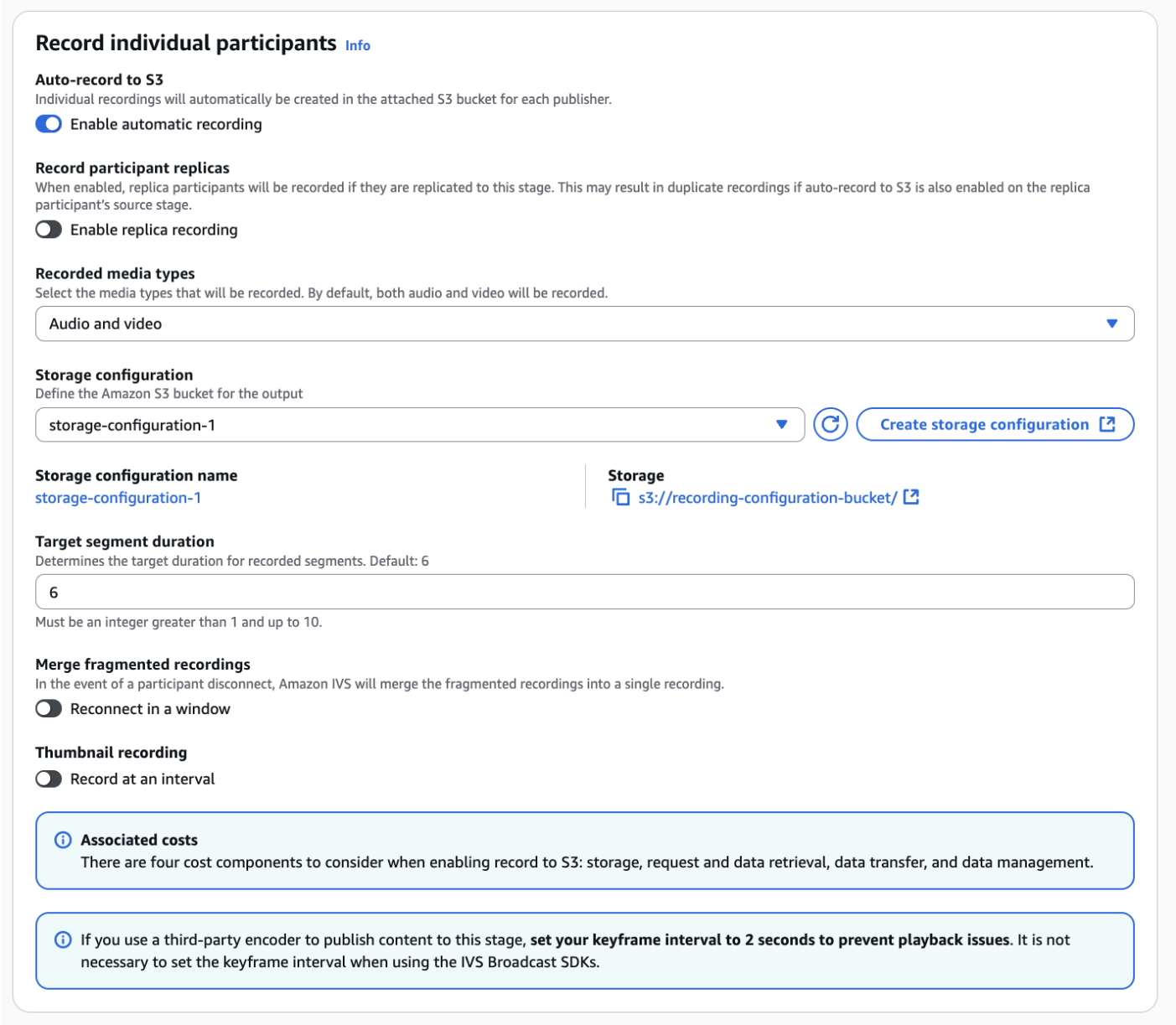


4. 次のフィールドに入力します。

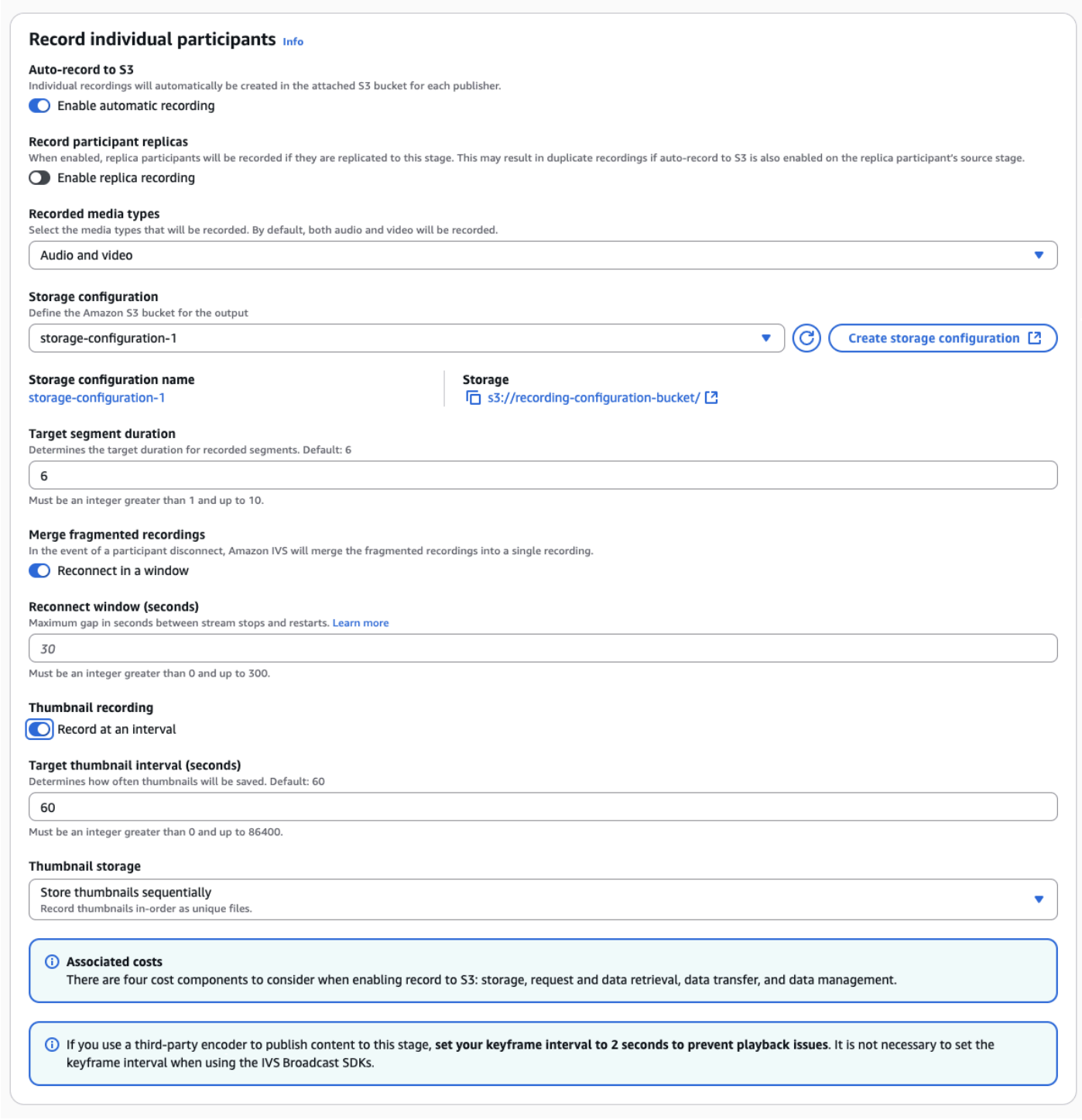
   a. 必要に応じて保存設定名を入力します。

   b. [バケット名] を入力します。

5. [保存設定を作成する] を選択し、一意の ARN を持つ新規の保存設定リソースを作成します。通常、録画設定の作成は数秒ですが、最大で 20 秒かかることがあります。保存設定が作成されると、[チャネルを作成する] ウィンドウに戻ります。その個々の参加者を記録するエリアには、新規の保存設定および作成した S3 バケット (ストレージ) が表示されます。



6. 必要に応じて、参加者レプリカの記録、個々の参加者による記録のマージ、サムネイル記録など、デフォルトでは無効になっているその他のオプションを有効にすることができます。



IVS ステージを作成するための CLI 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。

個々の参加者による記録の有効の有無を問わず、ステージを作成するかどうかにより、CLI を使用して以下の 2 手順のいずれかに従ってリソースを作成および管理できるようになりました。

個々の参加者の記録なしでステージを作成する

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"
   }
}
```

個々の参加者の記録でステージを作成する

個々の参加者の記録を有効にしてステージを作成する方法

```
aws ivs-realtime create-stage --name "test-stage-participant-recording" --auto-
participant-recording-configuration storageConfigurationArn=arn:aws:ivs:us-
west-2:123456789012:storage-configuration/LKZ6QR7r55c2,mediaTypes=AUDIO_VIDEO
```

必要に応じて thumbnailConfiguration パラメータを渡してサムネイルの保存、記録モード、サムネイル間隔秒数を手動で設定します。

```
aws ivs-realtime create-stage --name "test-stage-participant-recording" --auto-
participant-recording-configuration storageConfigurationArn=arn:aws:ivs:us-
west-2:123456789012:storage-configuration/
LKZ6QR7r55c2,mediaTypes=AUDIO_VIDEO,thumbnailConfiguration="{targetIntervalSeconds=10,storage=[
```

必要に応じて recordingReconnectWindowSeconds パラメータを渡し、フラグメント化された個々の参加者の記録をマージできるようにします。

```
aws ivs-realtime create-stage --name "test-stage-participant-recording" --auto-
participant-recording-configuration "storageConfigurationArn=arn:aws:ivs:us-
```

```
west-2:123456789012:storage-configuration/
LKZ6QR7r55c2,mediaTypes=AUDIO_VIDEO,thumbnailConfiguration="{targetIntervalSeconds=10,storage=[
```

レスポンスは次のとおりです。

```
{
   "stage": {
      "arn": "arn:aws:ivs:us-west-2:123456789012:stage/VSWjvX5XOkU3",
      "autoParticipantRecordingConfiguration": {
         "hlsConfiguration": {
             "targetSegmentDurationSeconds": 6
         },
         "mediaTypes": [
            "AUDIO_VIDEO"
         ],
         "recordingReconnectWindowSeconds": 60,
         "recordParticipantReplicas": true,
         "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/LKZ6QR7r55c2",
         "thumbnailConfiguration": {
            "recordingMode": "INTERVAL",
            "storage": [
               "SEQUENTIAL",
               "LATEST"
            ],
            "targetIntervalSeconds": 10
         }
      },
      "endpoints": {
         "events": "<events-endpoint>",
         "rtmp": "<rtmp-endpoint>",
         "rtmps": "<rtmps-endpoint>",
         "whip": "<whip-endpoint>"
      },
      "name": "test-stage-participant-recording"
   }
}
```

ステップ 3: 参加者トークンを配布する

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。トークンを生成するには、次の 2 つの方法があります。

- キーペアを使用してトークンを作成します。
- IVS リアルタイムストリーミング API を使用してトークンを作成します。

両方のアプローチを以下にて説明します。

キーペアを使用したトークンの作成

サーバーアプリケーションでトークンを作成し、参加者に配布してステージに参加できます。ECDSA パブリック/プライベートキーペアを生成して JWT に署名し、パブリックキーを IVS にインポートします。その後、IVS はステージ結合時にトークンを検証できます。

IVS にはキーの有効期限はありません。プライベートキーが侵害された場合は、古いパブリックキーを削除する必要があります。

新しいキーペアの作成

キーペアを作成するには、複数の方法があります。以下に 2 つの例を示します。

コンソールを使用して新しいキーペアを作成するには、次の手順に従います。

1. Amazon IVS コンソールを開きます。ステージのリージョンを選択していない場合は、リージョンを選択します。
2. 左側のナビゲーションメニューで、[リアルタイムストリーミング] > [パブリックキー]を選択します。
3. [パブリックキーを作成] を選択します。[パブリックキーの作成] ダイアログが表示されます。
4. プロンプトに従って、[Create (作成)] を選択します。
5. Amazon IVS が新しいキーペアを生成します。パブリックキーはパブリックキーリソースとしてインポートされ、プライベートキーはすぐにダウンロードできるようになります。パブリックキーは、必要に応じて後でダウンロードすることもできます。

   Amazon IVS はクライアント側でキーを生成し、プライベートキーを保存しません。キーは必ず保存してください。後で取得することはできません。

OpenSSL で P384 EC キーペアを作成するには (最初に OpenSSL のインストールが必要な場合があります)、次の手順を行います。このプロセスにより、プライベートキーとパブリックキーの両方にアクセスできます。パブリックキーは、トークンの検証をテストする場合にのみ必要です。

```
openssl ecparam -name secp384r1 -genkey -noout -out priv.pem
openssl ec -in priv.pem -pubout -out public.pem
```

次に、以下の手順に従い、新しいパブリックキーをインポートします。

パブリックキーをインポートするには

キーペアが作成できたら、パブリックキーを IVS にインポートできます。プライベートキーはシステムでは必要ありませんが、ユーザーがトークンを署名するのに使用されます。

コンソールで既存のパブリックキーをインポートするには

1. Amazon IVS コンソールを開きます。ステージのリージョンを選択していない場合は、リージョンを選択します。
2. 左側のナビゲーションメニューで、[リアルタイムストリーミング] > [パブリックキー]を選択します。
3. [インポート] を選択します。[パブリックキーのインポート] ダイアログが表示されます。
4. プロンプトに従って、[インポート] を選択します。
5. Amazon IVS はパブリックキーをインポートし、パブリックキーリソースを生成します。

CLI で既存のパブリックキーをインポートするには、次のコマンドを実行します。

```
aws ivs-realtime import-public-key --public-key-material "`cat public.pem`" --region
 <aws-region>
```

リージョンがローカルの AWS 設定ファイルにある場合、`--region <aws-region>` を省略できます。

レスポンスの例を次に示します。

```
{
    "publicKey": {
        "arn": "arn:aws:ivs:us-west-2:123456789012:public-key/f99cde61-c2b0-4df3-8941-
ca7d38acca1a",
        "fingerprint": "98:0d:1a:a0:19:96:1e:ea:0a:0a:2c:9a:42:19:2b:e7",
```

```
        "publicKeyMaterial": "-----BEGIN PUBLIC KEY-----
\nMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEVjYMV+P4ML6xemanCrtse/FDwsNnpYmS
\nS6vRV9Wx37mjwi02hObKuCJqpj7x0lpz0bHm5v1JBvdZYAd/r2LR5aChK+/GM2Wj
\nl8MG9NJIVFaw1u3bvjEjzTASSfS1BDX1\n-----END PUBLIC KEY-----\n",
        "tags": {}
    }
}
```

API リクエスト

```
POST /ImportPublicKey HTTP/1.1
{
  "publicKeyMaterial": "<pem file contents>"
}
```

トークンを生成して署名する

JWT およびトークン署名用にサポートされているライブラリの操作の詳細については、jwt.io を参照してください。jwt.io インターフェイスでは、プライベートキーを入力してトークンに署名する必要があります。パブリックキーは、トークンを検証する場合にのみ必要です。

すべての JWT には、ヘッダー、ペイロード、署名の 3 つのフィールドがあります。

JWT のヘッダーとペイロードの JSON スキーマを以下に示します。または、IVS コンソールからサンプル JSON をコピーすることもできます。IVS コンソールからヘッダーとペイロード JSON を取得するには

1. Amazon IVS コンソールを開きます。ステージのリージョンを選択していない場合は、リージョンを選択します。
2. 左側のナビゲーションメニューで、[リアルタイムストリーミング] > [ステージ]を選択します。
3. 使用するステージを選択します。[詳細を表示] 選択します。
4. [参加者トークン] セクションで、[トークンの作成] の横にあるドロップダウンを選択します。
5. [トークンヘッダーとペイロードをビルドする]を選択します。
6. フォームに入力し、ポップアップの下部に表示される JWT ヘッダーとペイロードをコピーします。

トークンスキーマ: ヘッダー

ヘッダーは以下を指定します。

- alg は署名アルゴリズムです。これは、SHA-384 ハッシュアルゴリズムを使用する ECDSA 署名アルゴリズムの ES384 です。
- typ はトークン型、JWT です。
- kid は、トークンの署名に使用されるパブリックキーの ARN です。GetPublicKey API リクエストから返されるのと同じ ARN である必要があります。

```
{
  "alg": "ES384",
  "typ": "JWT"
  “kid”: “arn:aws:ivs:123456789012:us-east-1:public-key/abcdefg12345”
}
```

トークンスキーマ: ペイロード

ペイロードには、IVS 固有のデータが含まれています。user_id を除くすべてのフィールドは必須です。

- JWT 仕様の RegisteredClaims は、ステージトークンを有効にするために提供する必要がある予約クレームです。
  - exp (有効期限) は、トークンの有効期限を示す Unix UTC タイムスタンプです。(Unix タイムスタンプは、うるう秒を無視した、1970-01-01T00:00:00Z UTC から、指定された UTC 日付/時刻までの秒数を表す数値です。) トークンは、参加者がステージに参加するときに検証されます。IVS は、デフォルト (かつ推奨値) の 12 時間 TTL を持つトークンを提供します。これは、公開時点 (iat) から最大 14 日間まで延長できます。整数型の値である必要があります。
  - iat (発行時) は、JWT が発行されたときの Unix UTC タイムスタンプです。(Unix タイムスタンプについては、exp のメモを参照してください。) 整数型の値である必要があります。
  - jti (JWT ID) は、トークンが付与される参加者を追跡および参照するために使用される参加者 ID です。すべてのトークンには一意の参加者 ID が必要です。英数字、ハイフン (-)、アンダースコア (_) 文字のみを含む、最大 64 文字の大文字と小文字を区別する文字列である必要があります。他の特殊文字は使用できません。
- user_id は、トークンを識別するのに役立つ、顧客に割り当てられたオプションの名前です。これは、参加者を顧客自身のシステム内のユーザーにリンクするために使用できます。これは、CreateParticipantToken API リクエストの userId フィールドと一致する必要があります。任意の UTF-8 エンコードテキストにすることができ、最大 128 文字の文字列です。このフィールドはすべてのステージ参加者に公開されます。これらを個人を特定する情報、機密情報や機微な情報には使用しないでください。

- resource はステージの ARN です。例: arn:aws:ivs:us-east-1:123456789012:stage/oRmLNwuCeMlQ。
- topic はステージの ID で、ステージ ARN から抽出できます。例えば、ステージ ARN が arn:aws:ivs:us-east-1:123456789012:stage/oRmLNwuCeMlQ の場合、ステージ ID は oRmLNwuCeMlQ です。
- events_url は、CreateStage または GetStage オペレーションから返されるイベントエンドポイントである必要があります。この値はステージ作成時にキャッシュすることをお勧めします。この値は最大 14 日間キャッシュできます。値の例は wss://global.events.live-video.net です。
- whip_url は、CreateStage または GetStage オペレーションから返される WHIP エンドポイントである必要があります。この値はステージ作成時にキャッシュすることをお勧めします。この値は最大 14 日間キャッシュできます。値の例は https://453fdfd2ad24df.global-bm.whip.live-video.net です。
- capabilities はトークンの機能を指定します。有効な値は allow_publish および allow_subscribe です。サブスクライブ専用トークンの場合は、allow_subscribe を true に設定するだけにします。
- attributes は、トークンにエンコードしてステージにアタッチするアプリケーション提供の属性を指定できるオプションのフィールドです。マップキーと値には、UTF-8 エンコードされたテキストを含めることができます。このフィールドの最大長は合計 1 KB です。このフィールドはすべてのステージ参加者に公開されます。これらを個人を特定する情報、機密情報や機微な情報には使用しないでください。
- version は 1.0 である必要があります。

```
{
  "exp": 1697322063,
  "iat": 1697149263,
  "jti": "Mx6clRRHODPy",
  "user_id": "<optional_customer_assigned_name>",
  "resource": "<stage_arn>",
  "topic": "<stage_id>",
  "events_url": "wss://global.events.live-video.net",
  "whip_url": "https://114ddfabadaf.global-bm.whip.live-video.net",
  "capabilities": {
    "allow_publish": true,
    "allow_subscribe": true
  },
  "attributes": {
    "optional_field_1": "abcd1234",
```

```
    "optional_field_2": "false"
  },
  "version": "1.0"
}
```

トークンスキーマ: 署名

署名を作成するには、ヘッダー (ES384) で指定されたアルゴリズムを使用して、エンコードされたヘッダーとエンコードされたペイロードに署名します。

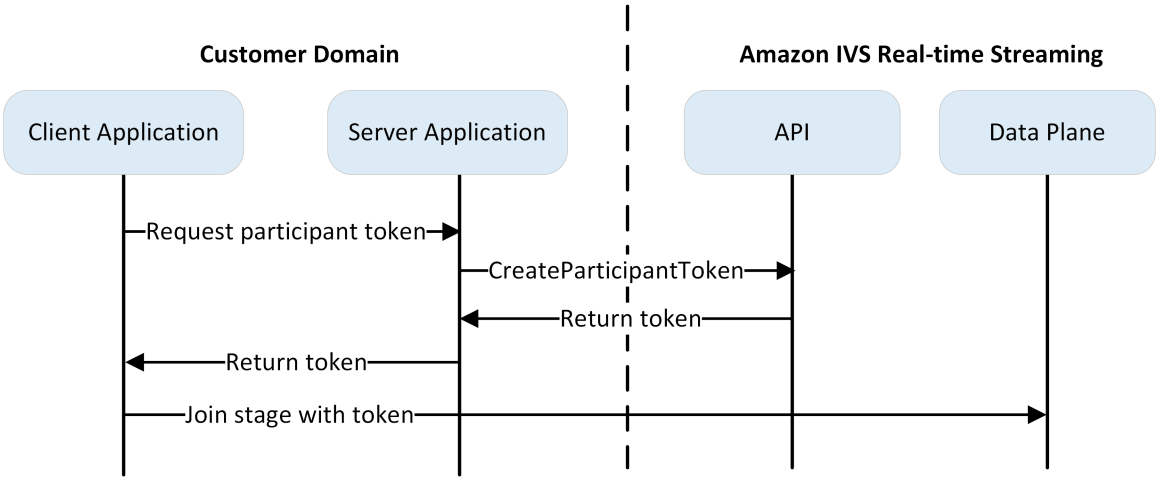
```
ECDSASHA384(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  <private-key>
)
```

指示

1. ES384 署名アルゴリズムと IVS に提供されるパブリックキーに関連付けられたプライベートキーを使用してトークンの署名を生成します。
2. トークンを組み立てます。

```
base64UrlEncode(header) + "." +
base64UrlEncode(payload) + "." +
base64UrlEncode(signature)
```

IVS Real-Time Streaming API を使用したトークンの作成



上記のように、クライアントアプリケーションからサーバー側のアプリケーションにトークンを要求し、サーバー側のアプリケーションは AWS SDK または SigV4 署名付きリクエストを使用して CreateParticipantToken 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は、オプションで属性や機能を指定できます。

- アプリケーションが提供する属性を指定してトークンにエンコードし、ステージにアタッチできます。マップキーと値には、UTF-8 エンコードされたテキストを含めることができます。このフィールドの最大長は合計 1 KB です。このフィールドはすべてのステージ参加者に公開されます。これらを個人を特定する情報、機密情報や機微な情報には使用しないでください。
- トークンで有効になっている機能を指定できます。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。

詳細については、「CreateParticipantToken」を参照してください。

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. [トークンの作成] を選択します。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. [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、JavaScript を使用する AWS SDK の手順です。

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 パッケージ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 JavaScript の開始」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);
```

ステップ 4: IVS Broadcast SDK を統合する

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。ステージに RTMP 取り込みを設定している場合は、RTMP エンドポイント (OBS や ffmpeg など) にブロードキャストできる任意のエンコーダを使用できます。

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。以下の手順では、BasicRealTime というアプリを作成する手順を説明します。アプリの全コードは CodePen と GitHub にあります。

- Web: https://codepen.io/amazon-ivs/pen/ZEqgrpo
- 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 ライブラリとして分散されており、https://web-broadcast.live-video.net/1.33.0/amazon-ivs-web-broadcast.js で入手できます。

`<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
```

これで IVSBroadcastClient 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:1.40.0: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

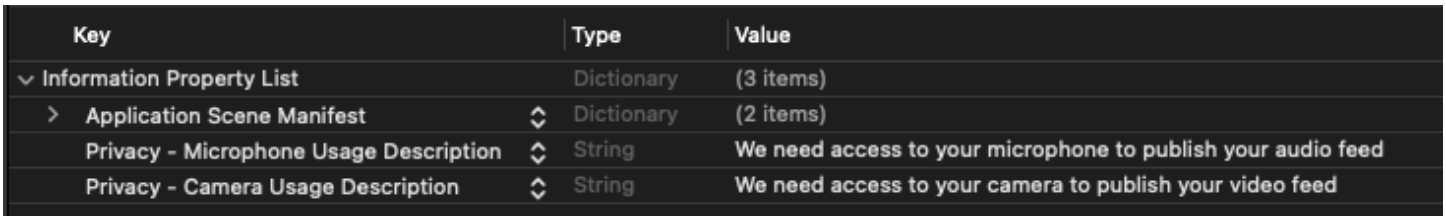
iOS プロジェクトの作成

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。手順については、「iOS Broadcast SDK Guide」の「ライブラリのインストール」を参照してください。

アクセス許可の設定

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。



ステップ 5: ビデオを公開およびサブスクライブする

IVS に対する (リアルタイムの) 公開/サブスクライブを、

- WebRTC および RTMPS をサポートする、ネイティブの IVS Broadcast SDK を使用して行えます。特に本稼働シナリオでは、こちらをお勧めします。Web、Android、および iOS の詳細については、以下を参照してください。
- Amazon IVS コンソール – ストリームのテストに適しています。以下の「」を参照してください。
- その他のストリーミングソフトウェアおよびハードウェアエンコーダー — RTMP、RTMPS、または WHIP プロトコルをサポートする任意のストリーミングエンコーダーを使用できます。詳細については、「ストリーミングの取り込み」を参照してください。

IVS コンソール

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、ハンバーガーアイコンを選択して展開します)
3. サブスクライブまたは公開するステージを選択して、詳細ページに移動します。
4. サブスクライブするには、ステージに 1 つ以上のパブリッシャーがある場合は、[サブスクライブ]タブのサブ[スクライブ]ボタンを押してサブスクライブできます。(これらのタブは、[一般的な設定] セクションの下にあります)
5. パブリッシュするには

a. [公開] タブを選択します。

b. カメラとマイクへの IVS コンソールアクセスを許可するように求められます。これらの権限を許可してください。

c. [公開] タブの下部にあるドロップダウンボックスを使用して、マイクとカメラの入力デバイスを選択します。

d. 公開を開始するには、[公開を開始] を選択します。

e. 公開されたコンテンツを表示するには、[サブスクライブ] タブに戻ります。

f. 公開を停止するには、[公開] タブに移動し、下部にある[公開を停止] ボタンを押します。

注: サブスクライブと公開はリソースを消費するため、ステージに接続した時間単位の料金が発生します。詳細については、IVS 料金表ページの「リアルタイムストリーミング」を参照してください。

IVS Web Broadcast SDK を使用してパブリッシュおよびサブスクライブする

このセクションでは、ウェブアプリケーションを使用してステージに発行およびサブスクライブする手順について説明します。

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/1.33.0/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。

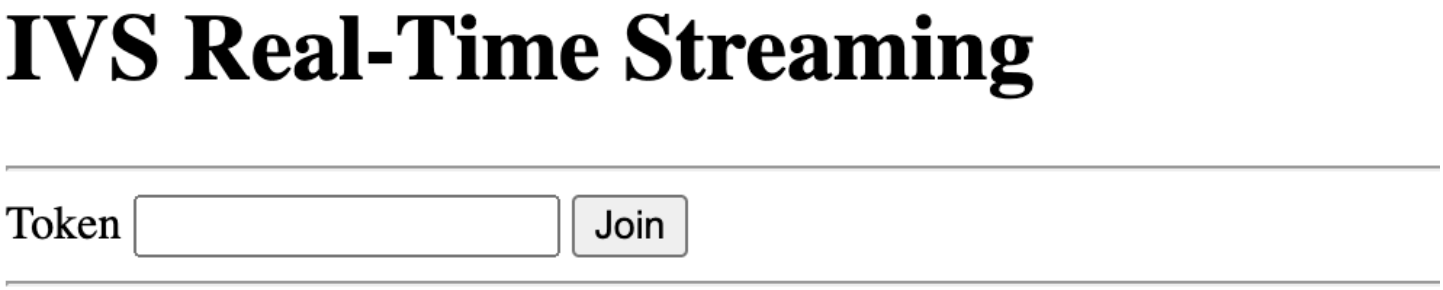
```
<h1>IVS Real-Time Streaming</h1>
<hr />

<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

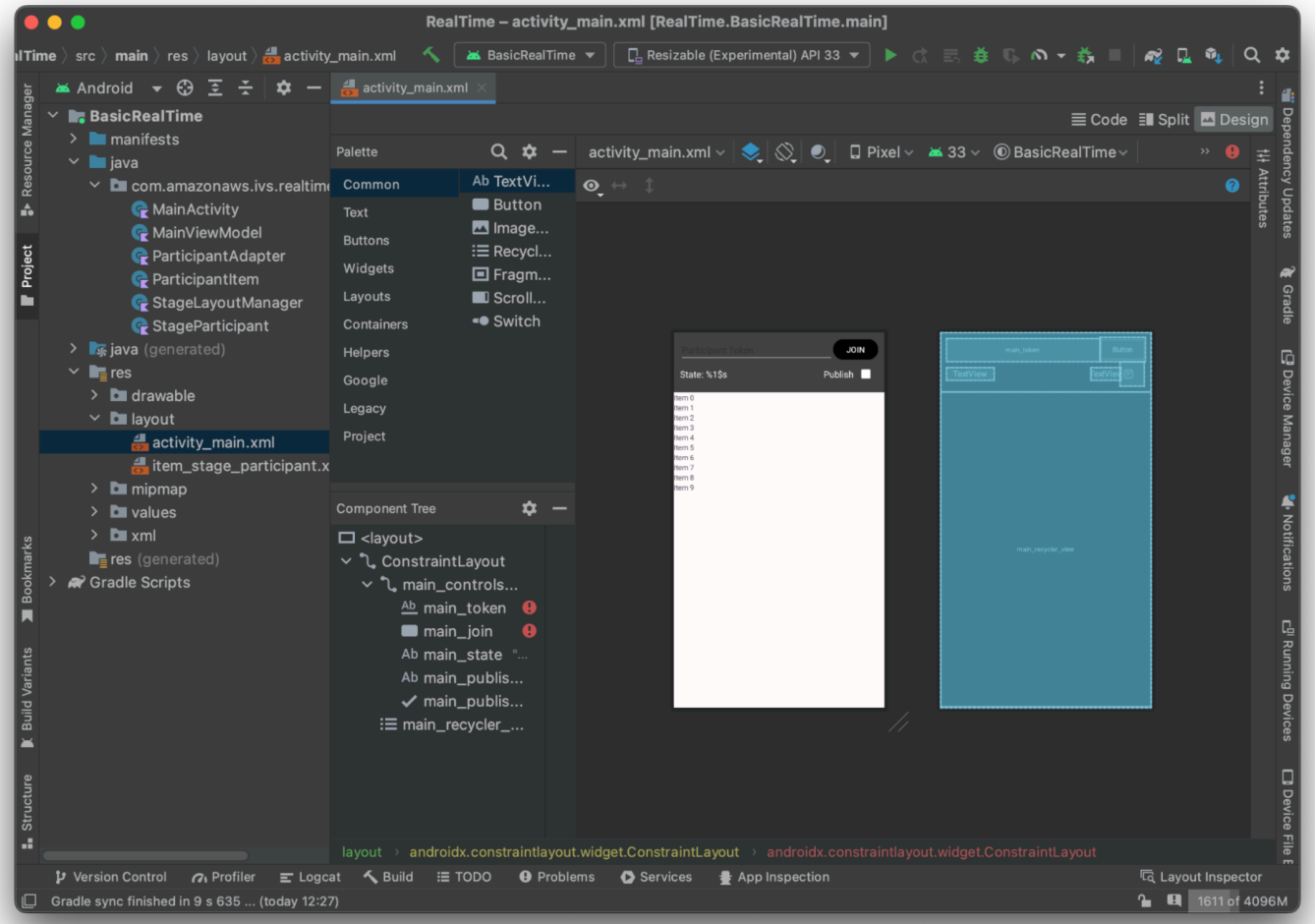
npm や React などに関する詳細な例については、「IVS Broadcast SDK: Web ガイド (リアルタイムストリーミングガイド」を参照してください。

IVS Android Broadcast SDK を使用してパブリッシュおよびサブスクライブする

このセクションでは、Android アプリケーションを使用してステージに発行およびサブスクライブする手順について説明します。

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>
```

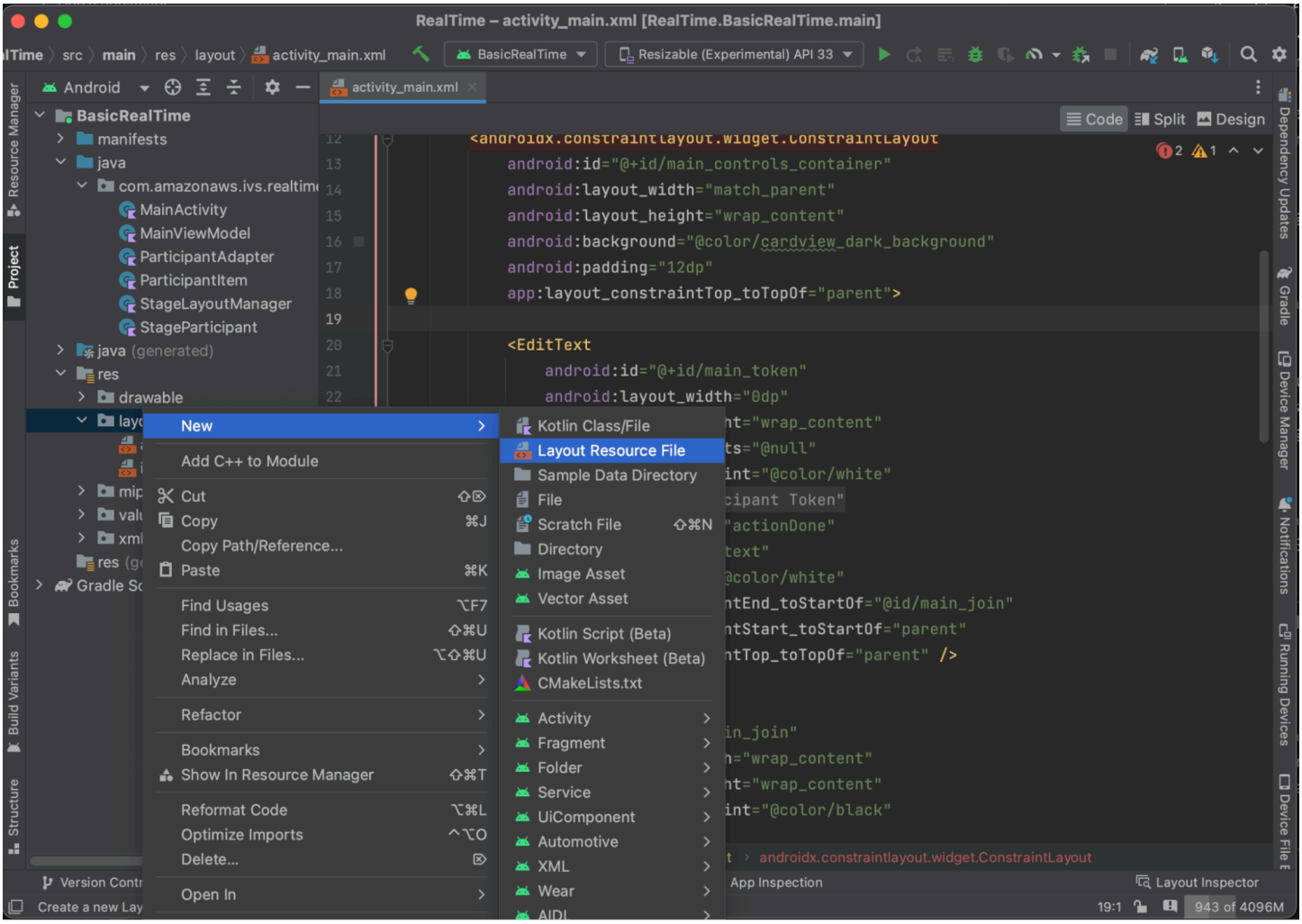
いくつかの文字列 ID を参照しました。全 `strings.xml` ファイルを作成しましょう。

```
<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>
```

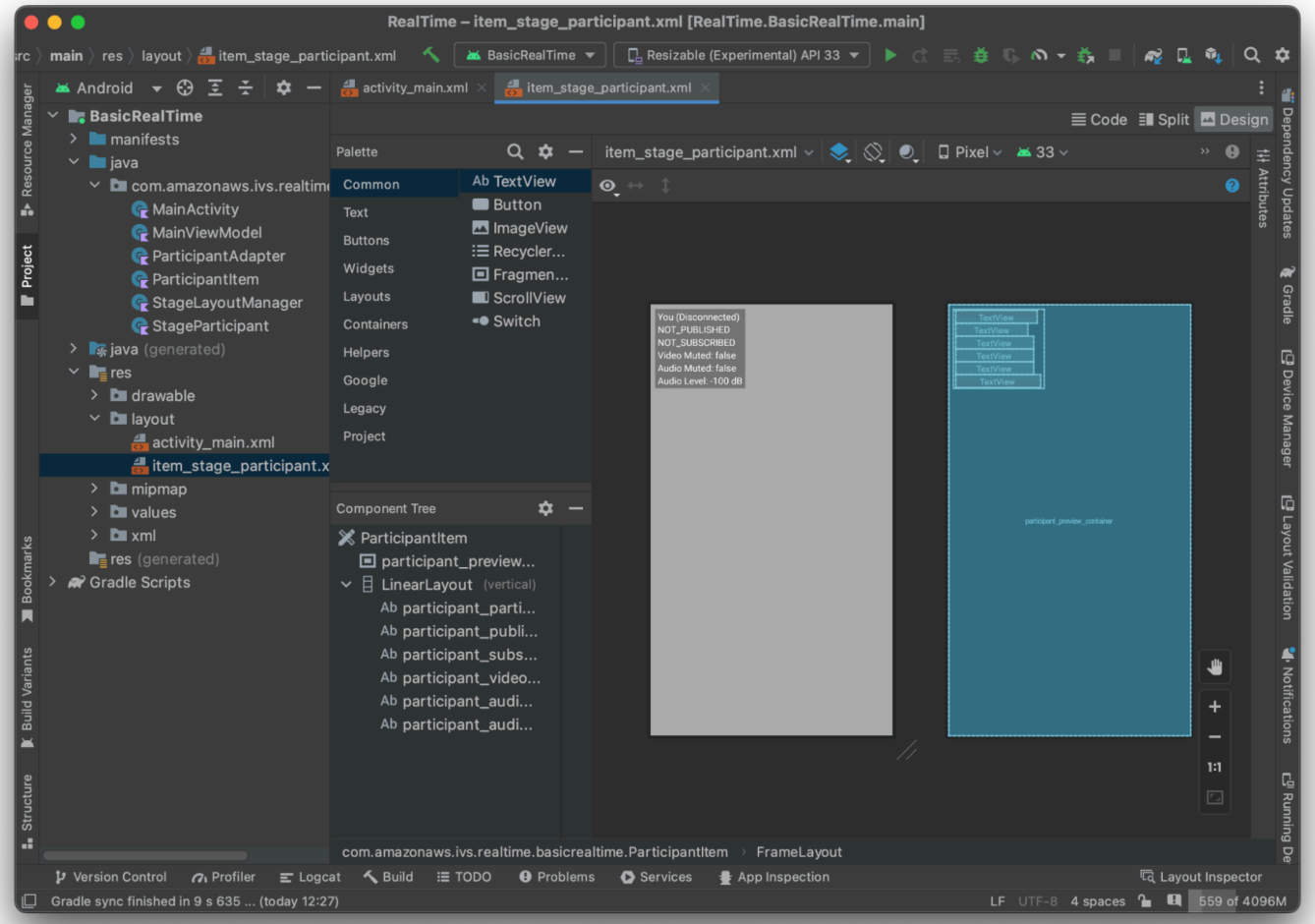
XML 内のこれらのビューを MainActivity.kt にリンクしましょう。

```
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

このアプリケーションは、MainViewModel.kt で参加者をローカルで追跡し、状態を Kotlin の StateFlow を使用して MainActivity に返します。

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 RecyclerView LayoutManager の実装

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
```

IVS iOS Broadcast SDK を使用してパブリッシュおよびサブスクライブする

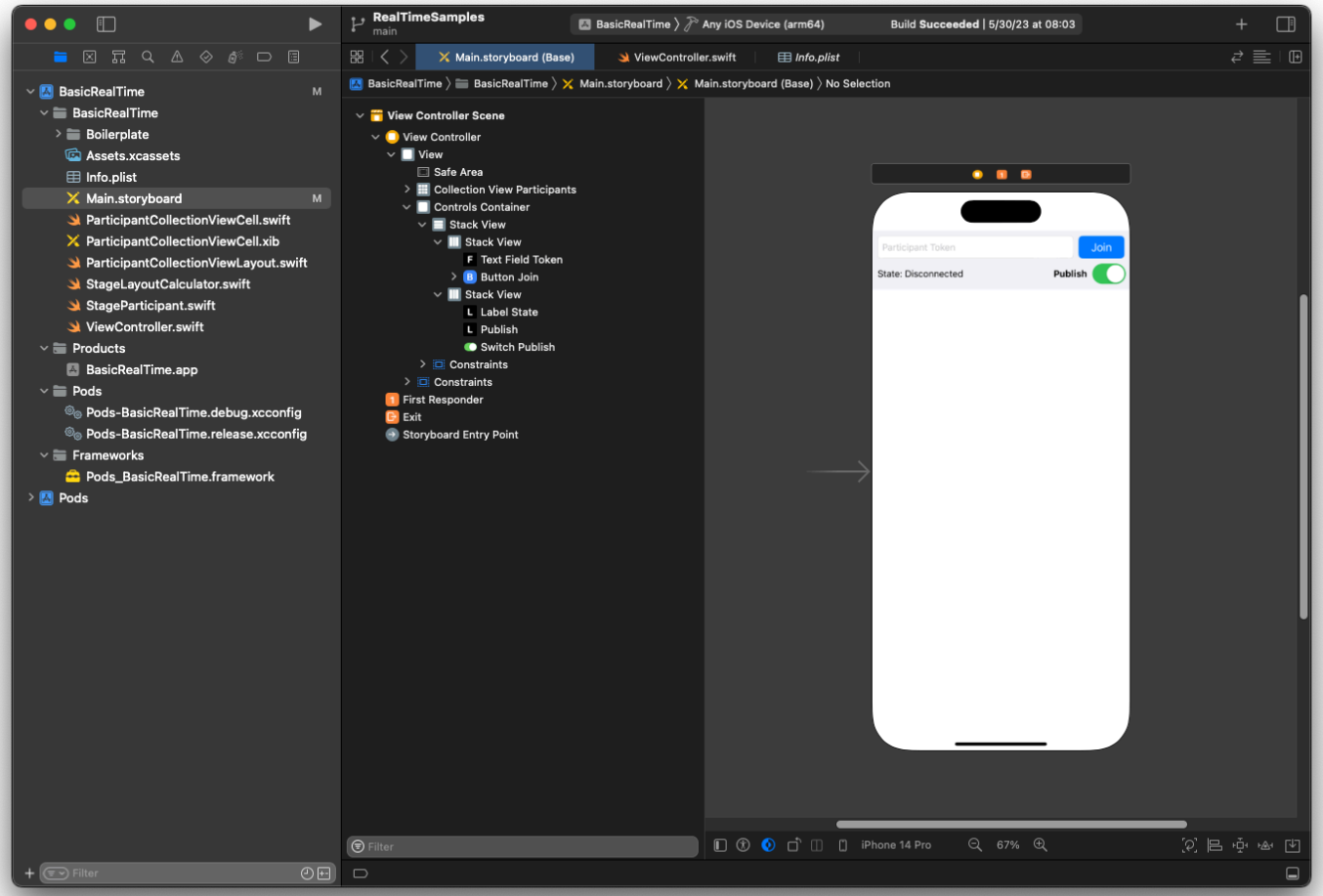
このセクションでは、iOS アプリケーションを使用してステージに発行およびサブスクライブする手順について説明します。

ビューの作成

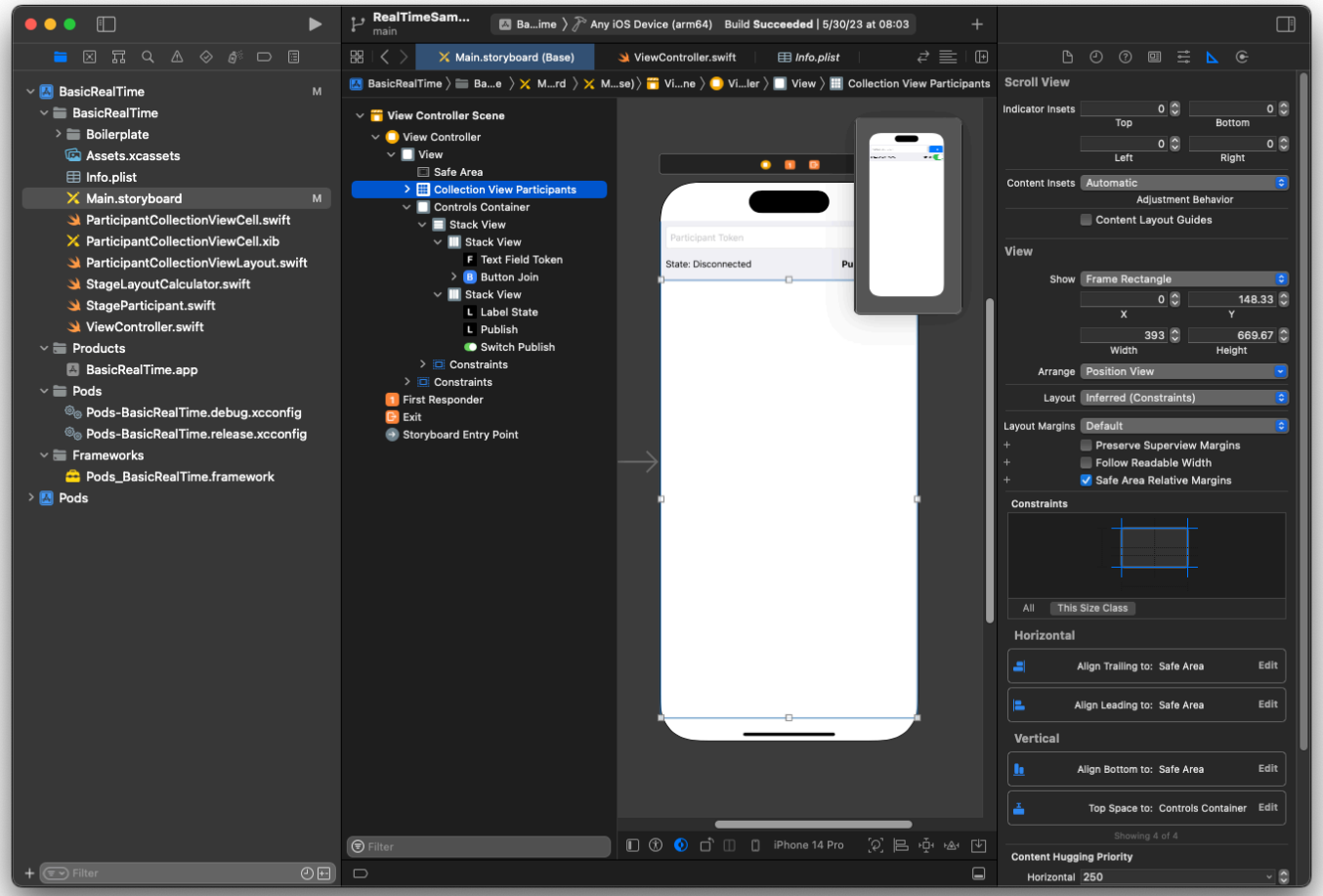
まず、自動作成された ViewController.swift ファイルを使用して AmazonIVSBroadcast をインポートし、リンクにいくつか @IBOutlets を追加します。

```
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!
```

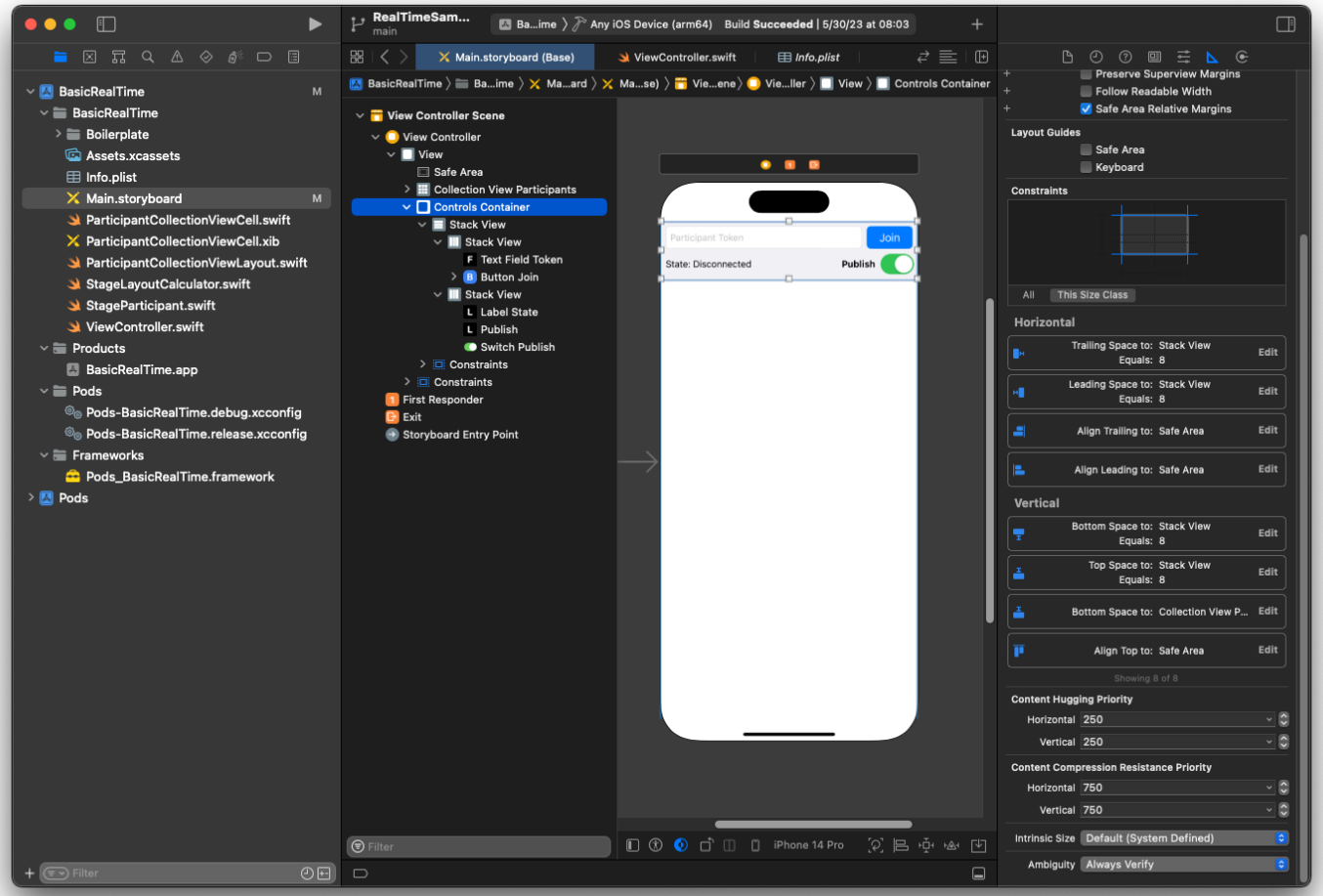
次に、これらのビューを作成して Main.storyboard でリンクします。使用するビュー構造は次のとおりです。



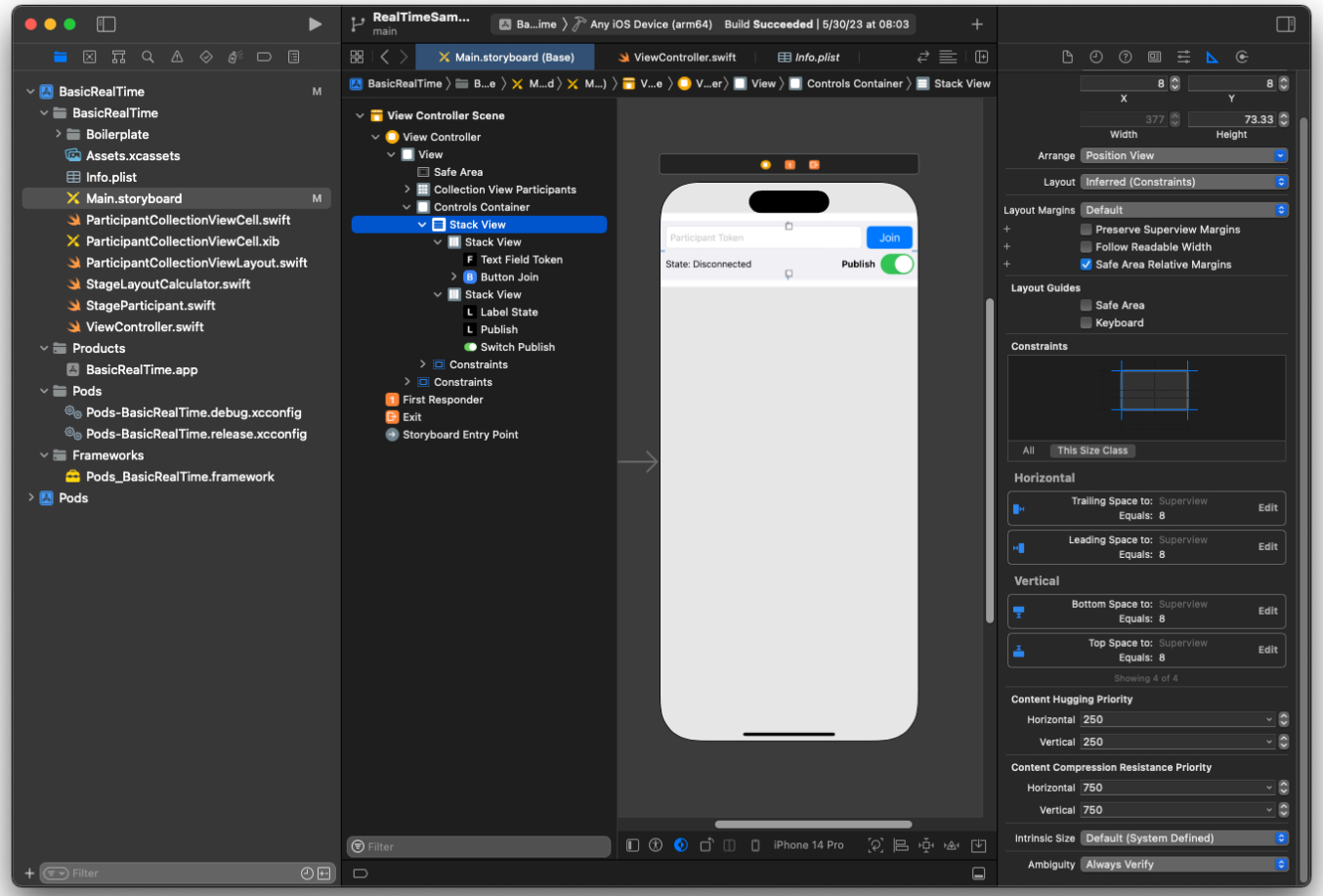
AutoLayout 設定では、3 つのビューをカスタマイズする必要があります。ズする必要があります。最初のビューは [Collection View Participants] (`UICollectionView`) です。[Leading]、[Trailing]、[Bottom] を [Safe Area] にバインドします。また、[Top] も [Controls Container] にバインドします。



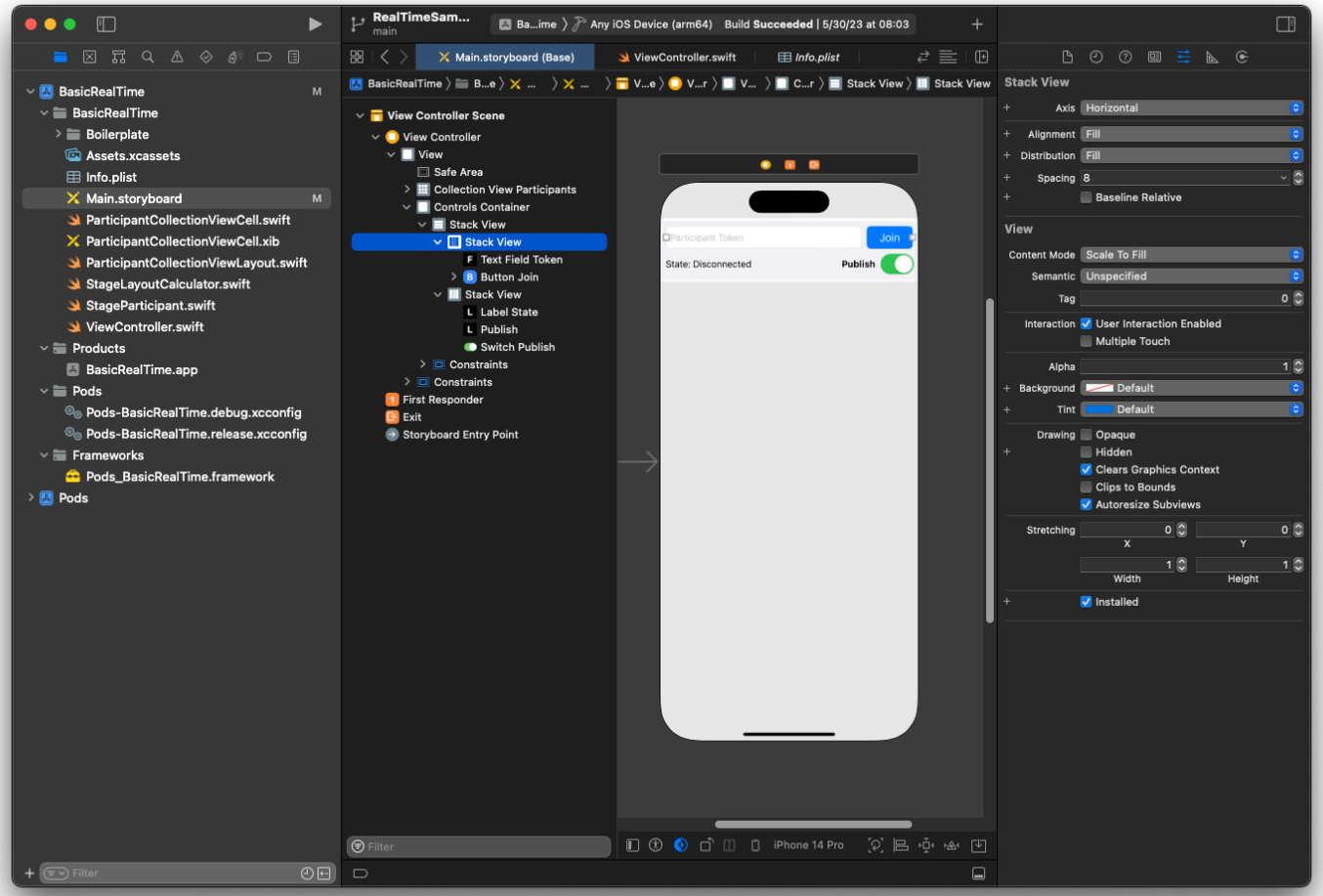
2 番目のビューは [コントロールコンテナ] です。[先頭]、[末尾]、[上] を [安全エリア] にバインドしました。



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。



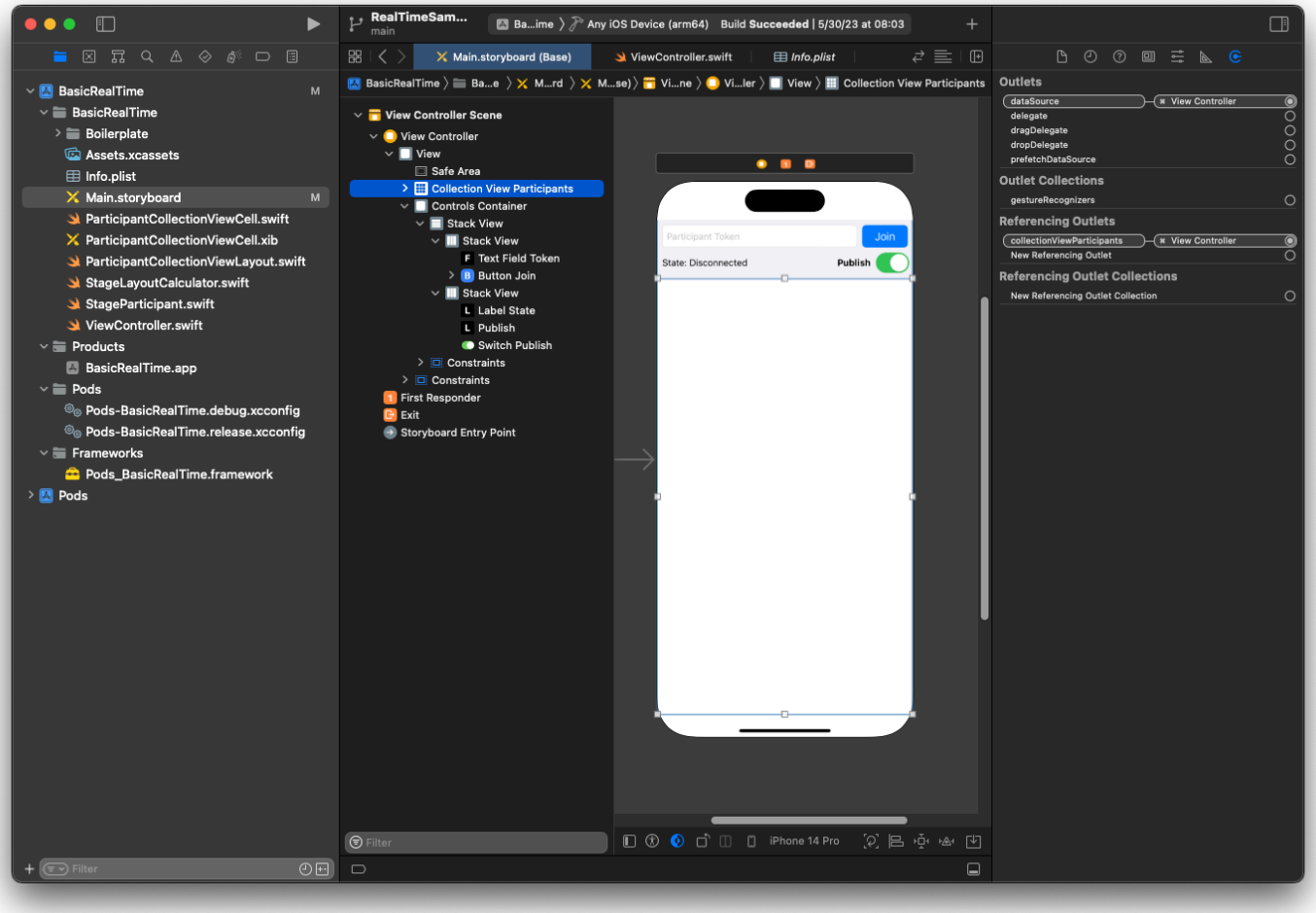
[UIStackViews] が残りのビューのレイアウトを処理します。3 つの [UIStackViews] すべてに対して、[配置] と [配信] には [入力] を使用します。



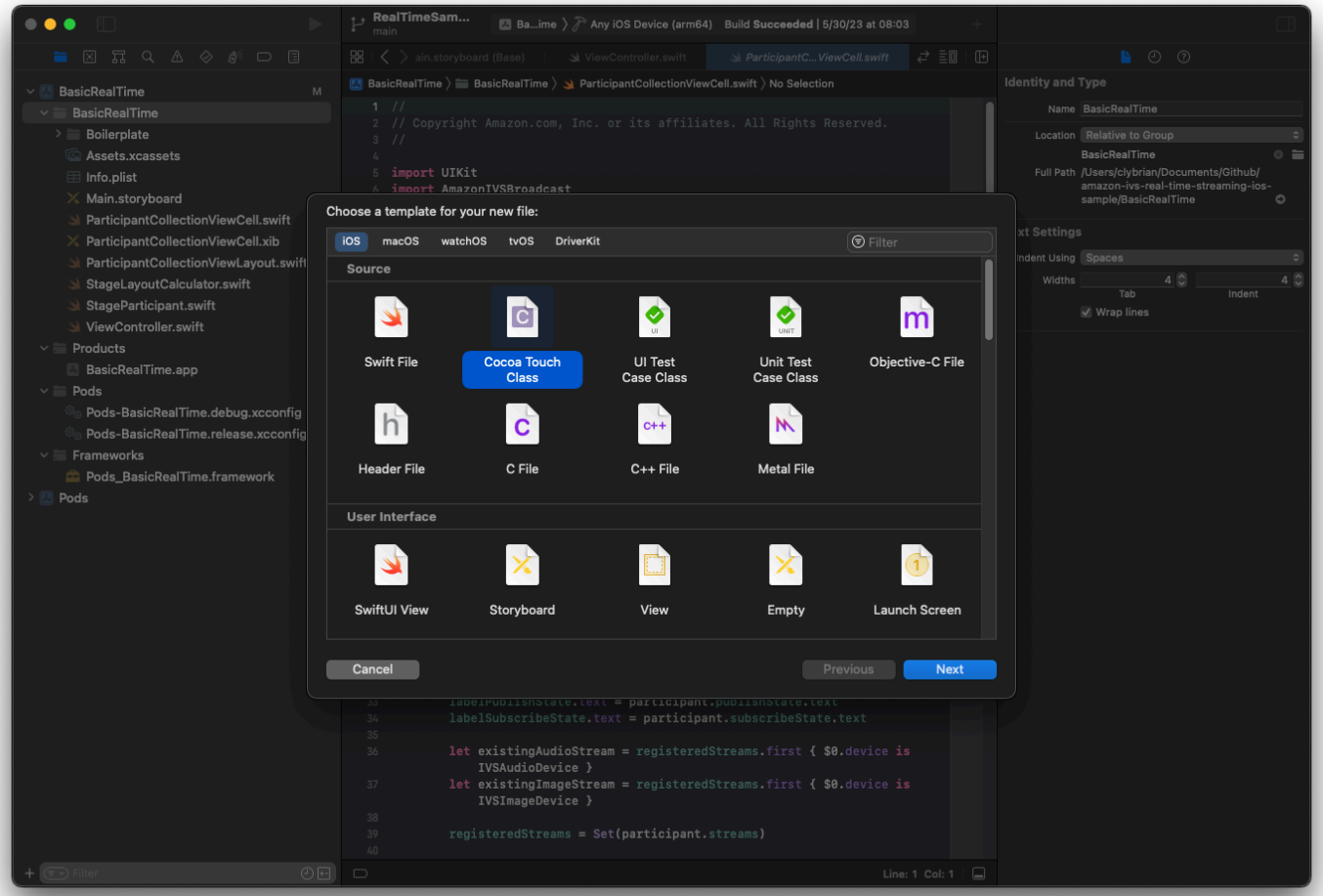
最後に、これらのビューを ViewController にリンクしましょう。上から、次のビューをマッピングします。

- [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。



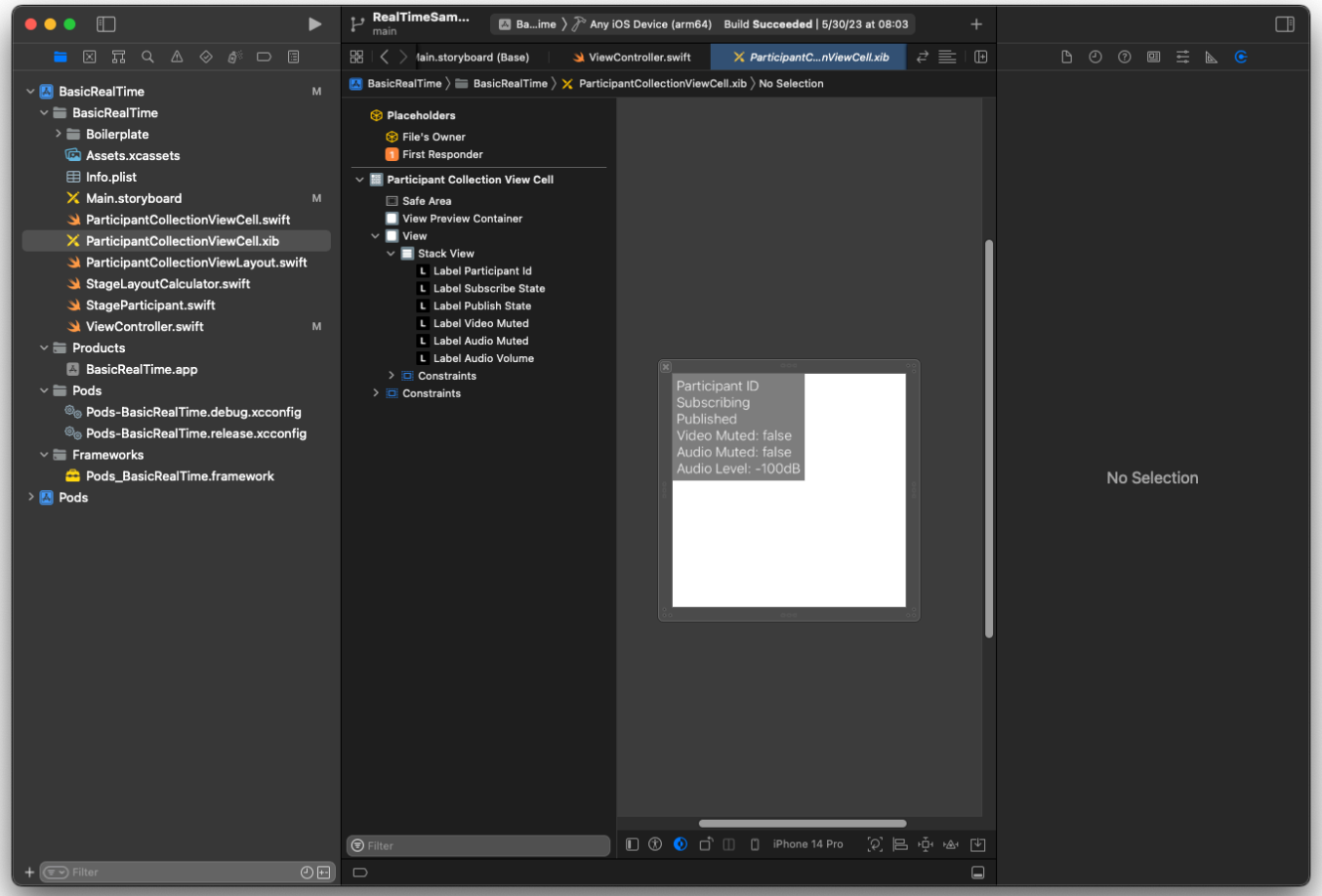
次に、参加者をレンダリングする `UICollectionViewCell` サブクラスを作成します。まず、新しい[Cocoa Touch Class] ファイルを作成します。



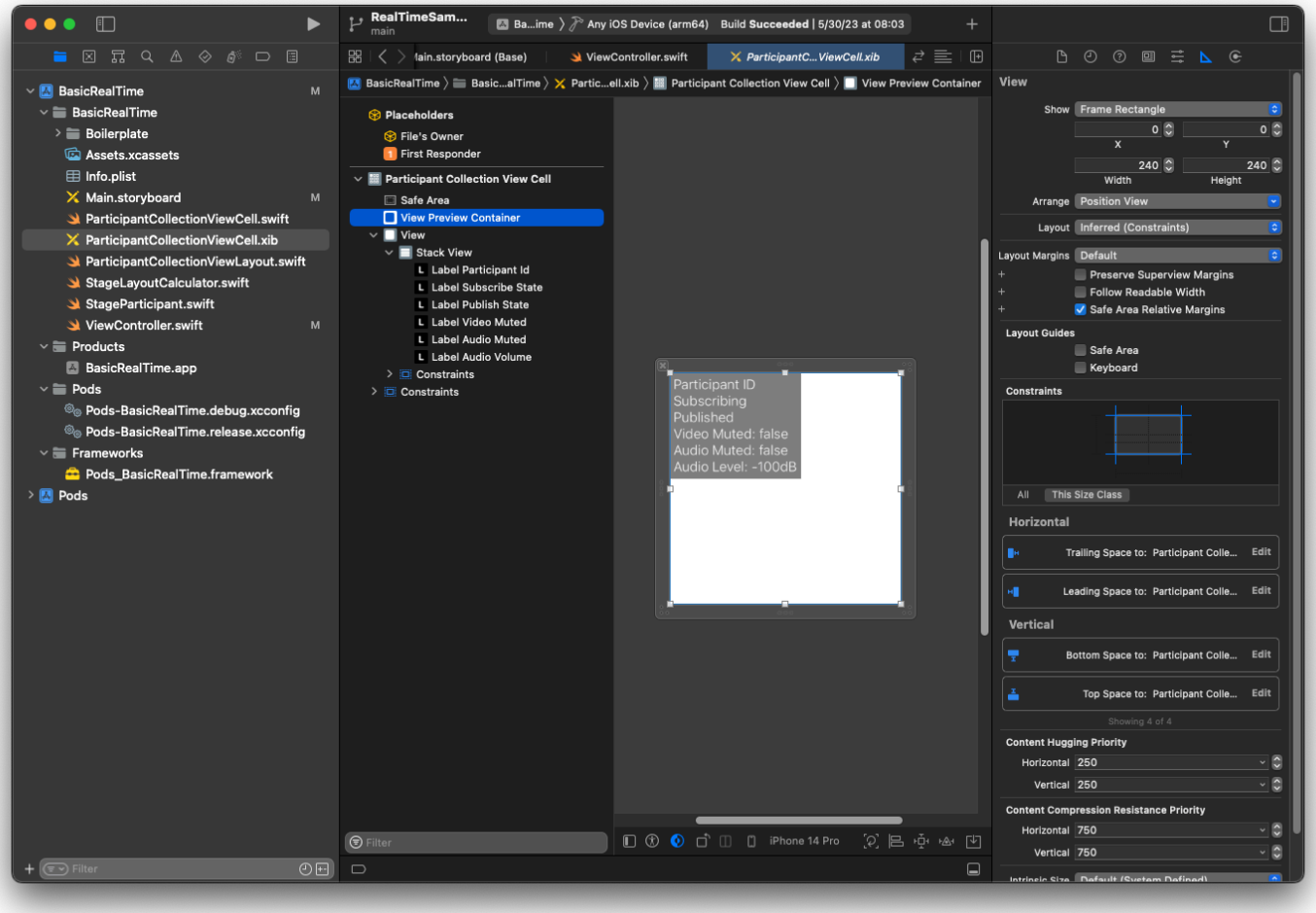
ParticipantUICollectionViewCell と名前を付けて、Swift 内にある UICollectionViewCell のサブクラスにします。もう一度、Swift ファイルから始めます。リンクする @IBOutlets を作成します。

```
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!
```

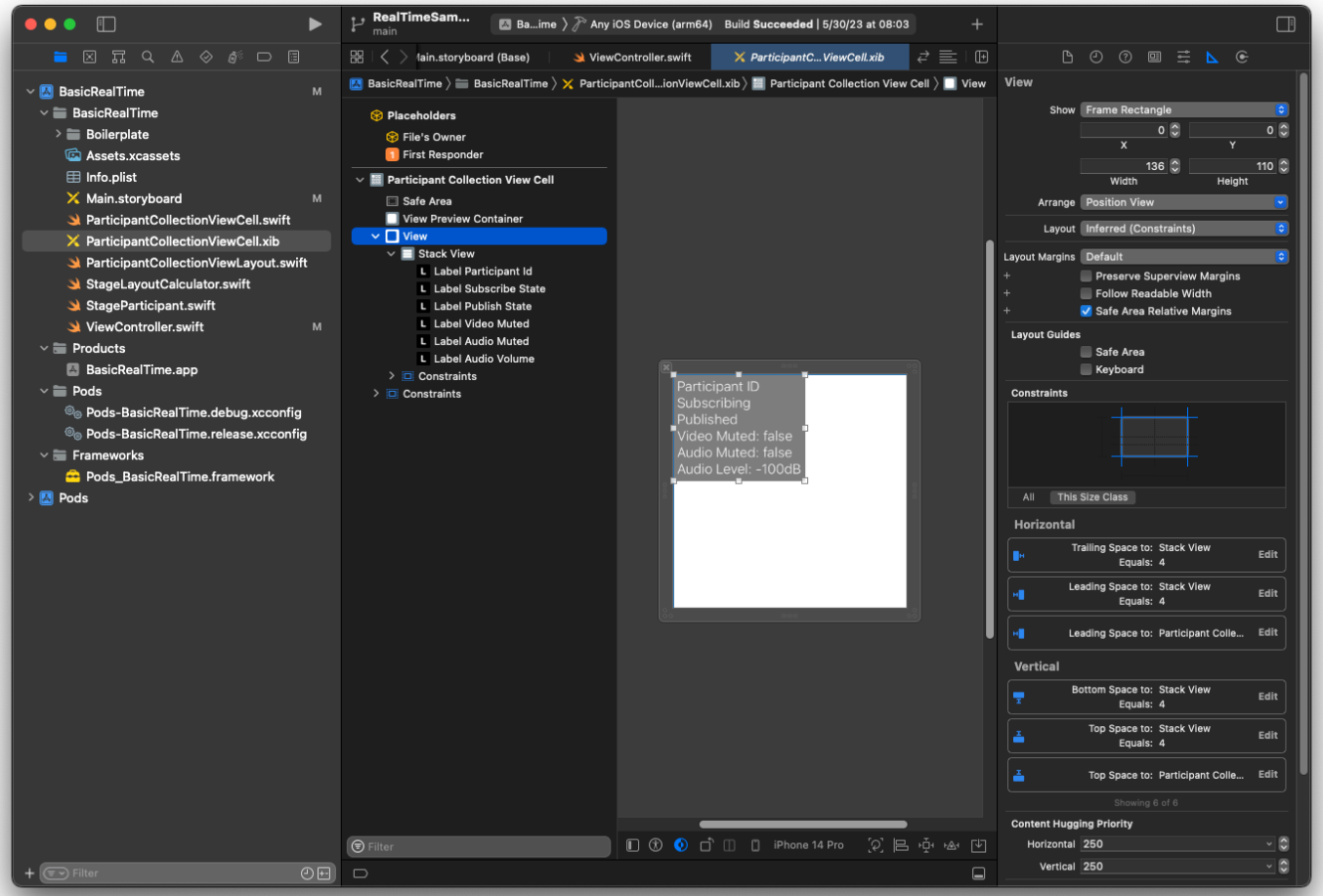
関連付けられている XIB ファイルに、次のビュー階層を作成します。



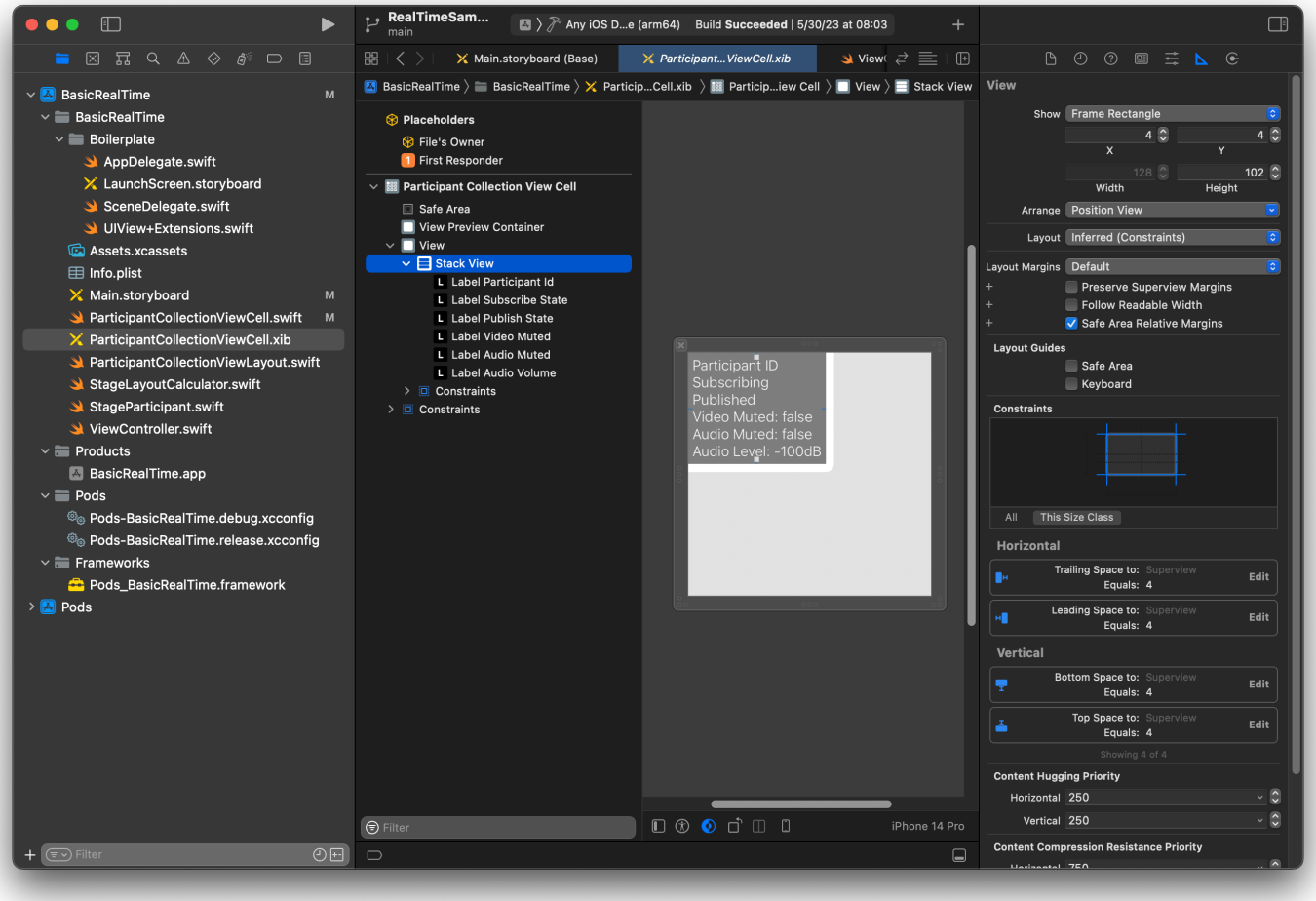
AutoLayout では、もう一度 3 つのビューを変更します。最初のビューは [ビュープレビューコンテナ] です。[末尾]、[先頭]、[上]、[下] を [参加者コレクションビューセル] に設定します。



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 [スーパービュー] 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 UICollectionViewLayout の実装

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。

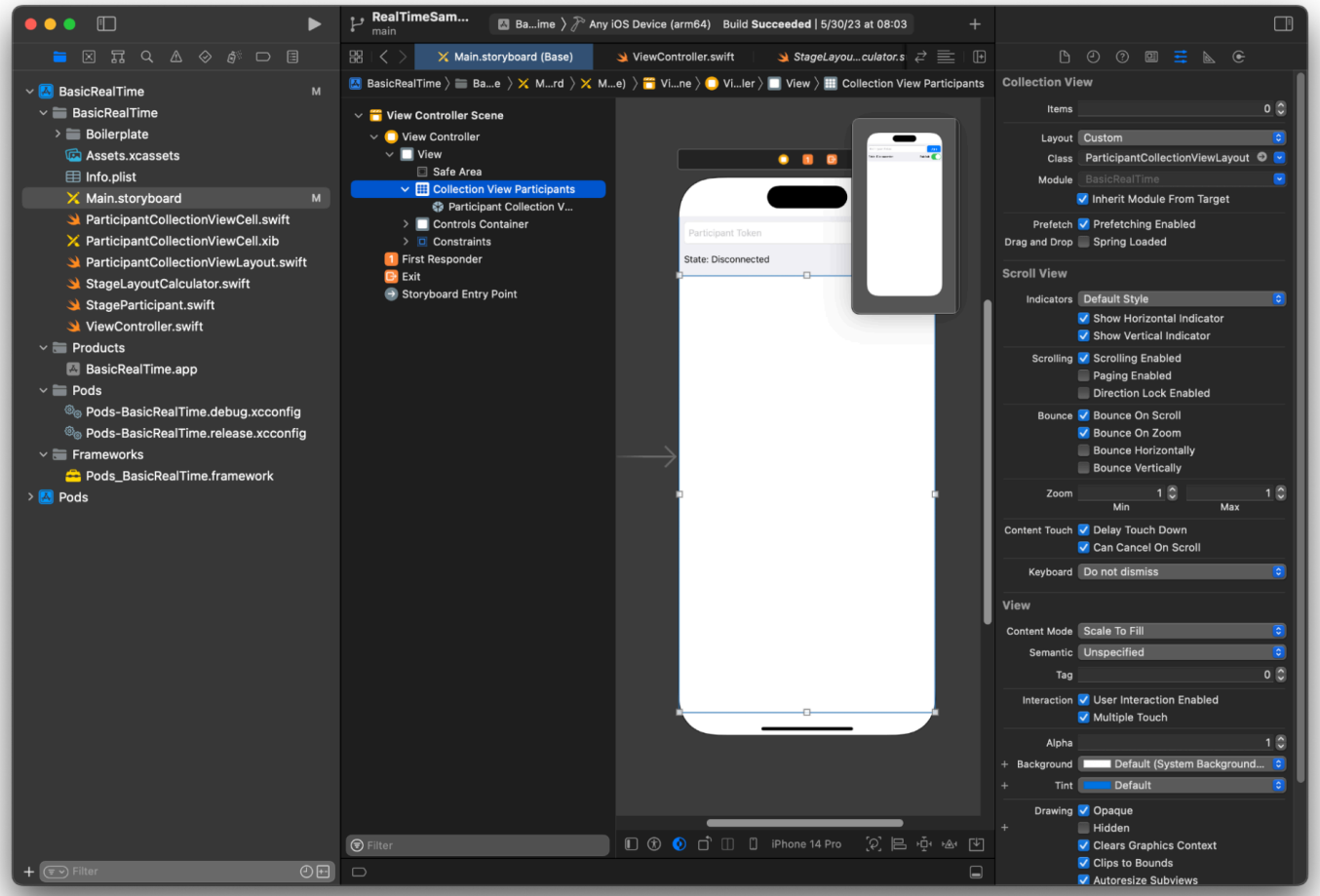
```
import Foundation
import UIKit
```

```
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

このドキュメントでは、IVS Real-Time Streaming アプリケーションのモニタリングに使用できるオプションについて詳しく説明します。

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 コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
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 コマンドラインインターフェイス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 Real-Time Streaming

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 none で、audio の MediaType では none 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。

参加者のレプリケーションの場合、宛先ステージにおいては、既存のステージヘルスメトリクスにすべてのレプリケートされた参加者 (ソースステージでのパブリッシャーであり、宛先ステージではレプリカ参加者となる) が含まれます。

メトリクス	ディメンション	説明
ConcurrentPublishers	—	AWS リージョンのすべてのステージで公開している参加者の数。 単位: 数 有効な統計: 平均、最大、最小
ConcurrentSubscriptions	—	AWS リージョンのすべてのステージにおけるパブリッシャーからサブスクライバーへの同時接続の数。 単位: 数 有効な統計: 平均、最大、最小
DownloadPacketLoss	—	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。 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	Platform	サブスクライバープラットフォームで DownloadPacketLoss をフィルタリングします。 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。

メトリクス	ディメンション	説明
DownloadPacketLoss	Platform, SDKVersion	サブスクライバープラットフォームと SDK バージョンで DownloadPacketLoss をフィルタリングします。 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	サブスクライバーステージで DownloadPacketLoss をフィルタリングします。 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。
DownloadPacketLoss	Stage, Platform	サブスクライバーステージとプラットフォームで DownloadPacketLoss をフィルタリングします。 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。

メトリクス	ディメンション	説明
DownloadPacketLoss	Stage, Platform, SDKVersion	サブスクライバーステージ、プラットフォーム、SDK バージョンで DownloadPacketLoss をフィルタリングします。 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, SubscriberCountryCode	サブスクライバーステージと国コード (ISO 3166) で DownloadPacketLoss をフィルタリングします。 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	SubscriberCountryCode	サブスクライバーの国コード (ISO 3166) で DownloadPacketLoss をフィルタリングします。 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。
DroppedFrames	–	サブスクライバーの場合: 受信フレームとサブスクライバーがサブスクライブしているすべてのパブリッシャー間でドロップされたフレームを集計して算出される、ビデオフレームのドロップ率。 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。

メトリクス	ディメンション	説明
DroppedFrames	Platform	サブスクライバーのプラットフォームで DroppedFrames をフィルタリングします。 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	Platform, SDKVersion	サブスクライバーのプラットフォームと SDK バージョンで DroppedFrames をフィルタリングします。 割合 (%) 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	ステージで DroppedFrames をフィルタリングします。 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	ステージと参加者で DroppedFrames をフィルタリングします。パブリッシャーでもあるサブスクライバーに対してのみ出力されます。 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。

メトリクス	ディメンション	説明
DroppedFrames	Stage, Platform	ステージとサブスクライバーのプラットフォームで DroppedFrames をフィルタリングします。 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, Platform, SDKVersion	ステージ、サブスクライバーのプラットフォーム、SDK バージョンで DroppedFrames をフィルタリングします。 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, SubscriberCountryCode	ステージとサブスクライバーの国コードで DroppedFrames をフィルタリングします。 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	SubscriberCountryCode	サブスクライバーの国別に DroppedFrames をフィルタリングします。 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。

メトリクス	ディメンション	説明
PublishBitrate	–	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。これには再送信されたデータが含まれます。ビットレートは、パケット消失や再送信によって膨らむ可能性があります。これは、パブリッシャーが送信する内容を反映しており、IVSが受信またはサブスクライバーに配信する内容と一致しない可能性があるためです。 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	Platform	パブリッシャーのプラットフォームで PublishBitrate をフィルタリングします。 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	ステージで PublishBitrate をフィルタリングします。 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。

メトリクス	ディメンション	説明
PublishBitrate	Stage, Participant, SimulcastLayer, MediaType	ステージ、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: 数 有効な統計: 平均、最大、最小
PublishFramerate	Stage, Participant	特定のパブリッシャーからビデオフレームを受信する頻度。このメトリクスは、RTMP 経由で公開する参加者のみが使用できます。 単位: カウント/秒 有効な統計: 平均、最大、最小 — 設定された間隔でのフレームの平均数、最大数、または最小数。
PublishFramerate	Stage, Participant, SimulcastLayer, MediaType	特定のパブリッシャーからビデオフレームを受信する頻度。このメトリクスは、RTMP 経由で公開する参加者のみが使用できます。 単位: カウント/秒 有効な統計: 平均、最大、最小 — 設定された間隔でのフレームの平均数、最大数、または最小数。

メトリクス	ディメンション	説明
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: 数 有効な統計: 平均、最大、最小
SubscribeBitrate	–	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	Platform	サブスクライバーのプラットフォームで SubscribeBitrate をフィルタリングします。 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	Platform, SDKVersion	サブスクライバーのプラットフォームと SDK バージョンで SubscribeBitrate をフィルタリングします。 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。

メトリクス	ディメンション	説明
SubscribeBitrate	Stage	ステージで SubscribeBitrate をフィルタリングします。 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	ステージ、参加者、メディアタイプで SubscribeBitrate をフィルタリングします。メディアタイプは「ビデオ」または「オーディオ」です。メトリクスは、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。
SubscribeBitrate	Stage, Platform	ステージとサブスクライバーのプラットフォームで SubscribeBitrate をフィルタリングします。 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, Platform, SDKVersion	ステージ、サブスクライバーのプラットフォーム、SDK バージョンで SubscribeBitrate をフィルタリングします。 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
SubscribeBitrate	Stage, SubscriberCountryCode	ステージ、サブスクライバーの国コードで SubscribeBitrate をフィルタリングします。 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	SubscriberCountryCode	サブスクライバーの国コード (ISO 3166-1 alpha-2) で SubscribeBitrate をフィルタリングします。 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。
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: 数 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 6.0 移行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

- 動画のアーティファクトやブラックスクリーンの原因となるパフォーマンスの制約により、いずれのモバイルウェブブラウザでも、パブリッシャーで同時にパブリッシュ/サブスクライブする場合は、その数を 3 個までにするようお勧めします。パブリッシャーの数をそれより増やす必要がある場合は、オーディオのみのパブリッシュ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 Real-Time Streaming 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 Real-Time Streaming 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配信する
- 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
- ステージに配信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: 1.33.0 (リリースノート)

リファレンスドキュメント: Amazon IVS Web Broadcast SDK で利用できる最も重要なメソッドについては、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。

- 簡易再生
- 簡易配信とサブスクライブ
- 包括的な React リアルタイムコラボレーションデモ

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。

注: ブラウザからの公開は、追加のソフトウェアのインストールを必要としないため、エンドユーザーにとって便利です。ただし、ブラウザベースの公開は、ブラウザ環境の制約と変動の影響を受けます。安定性を優先する必要がある場合 (イベントストリーミングなど) 、通常はシステムリソースに直接アクセスでき、ブラウザの制限を回避できる、ブラウザ以外のソース (OBS Studioやその他の専用エンコーダーなど) から公開することをお勧めします。ブラウザ以外の公開オプションの詳細については、「ストリームの取り込み」ドキュメントを参照してください。

IVS Web Broadcast SDK の開始方法 | Real-Time Streaming

このドキュメントでは、IVS Real-Time Streaming Web Broadcast SDK の使用を開始するためのステップについて説明します。

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

Web Broadcast SDK は JavaScript ライブラリとして分散されており、https://web-broadcast.live-video.net/1.33.0/amazon-ivs-web-broadcast.js で入手できます。

以下の例で定義されているクラスおよび列挙型はグローバルオブジェクト `IVSBroadcastClient` にあります。

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

npmを使う

npm パッケージをインストールするには:

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

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'
```

サーバーサイドレンダリングのサポート

Web Broadcast SDK ステージライブラリは、ロード時にライブラリの機能に必要なブラウザプリミティブを参照するため、サーバー側のコンテキストにロードできません。これを回避するには、「Next と React を使用した Web Broadcast デモ」で示されているように、ライブラリを動的にロードします。

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「MediaDevices.getUserMedia()」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

キャプチャ可能なデバイスを確認するには、ブラウザの MediaDevices.enumerateDevices() メソッドにクエリを実行します。

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

デバイスから MediaStream 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では、メディアが 720p 解像度に制限されていることを確認します。具体的には、幅と高さの getUserMedia と getDisplayMedia の制約値は、乗算時に 921600 (1280 x 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 },
```

```
});
```

IVS Web Broadcast SDK での配信とサブスクライブ | Real-Time Streaming

このドキュメントでは、IVS Real-Time Streaming Web Broadcast SDK を使用してステージに配信とサブスクライブを行うためのステップについて説明します。

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配信し、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配信の状態、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配信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。例えば、CreateParticipantToken でトークンを作成するときに、アプリケーションが `role` 属性を提供しているとします。アプリケーションは、`StageParticipantInfo` 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。

```
const strategy = {

   shouldSubscribeToParticipant(participant) {
      switch (participant.attributes.role) {
```

```
            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。

参加者へのサブスクライブの設定

```
subscribeConfiguration(participant: StageParticipantInfo): SubscribeConfiguration
```

リモート参加者がサブスクライブしている場合 (「参加者へのサブスクライブ」を参照)、SDK はホストアプリケーションにその参加者のカスタムサブスクライブ設定についてクエリします。この設定はオプションであり、ホストアプリケーションがサブスクライバーの動作の特定の側面を制御できるようにします。設定できる内容の詳細については、SDK リファレンスドキュメントの「SubscribeConfiguration」を参照してください。

次に示すのは実装の例です。

```
const strategy = {

   subscribeConfiguration: (participant) => {
      return {
         jitterBuffer: {
            minDelay: JitterBufferMinDelay.MEDIUM
         }
      }

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

この実装では、サブスクライブしたすべての参加者のジッターバッファ最小遅延を MEDIUM のプリセットに更新します。

shouldSubscribeToParticipant を使用した、より高度な実装も可能です。指定された ParticipantInfo を使用して、特定の参加者のサブスクライブ設定を選択的に更新できます。

デフォルトの動作を使用することをお勧めします。カスタム設定は、特定の動作を変更したい場合にのみ指定します。

配信

```
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、特定の参加者を配信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配信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、ユーザーは常に配信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。(配信/配信停止も使用できますが、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。)

配信するストリームの選択

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

配信時には、これを使用して配信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配信」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、エンドユーザーがボタンをタップするまでホストアプリケーショ

ンが配信したくない場合、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、配信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。

Events

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_ADAPTION_CHANGED, (participant, stream, isAdapting)
 => ())
stage.on(StageEvents.STAGE_STREAM_LAYERS_CHANGED, (participant, stream, layers) => ())
stage.on(StageEvents.STAGE_STREAM_LAYER_SELECTED, (participant, stream, layer, reason)
 => ())
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {})
stage.on(StageEvents.STAGE_STREAM_SEI_MESSAGE_RECEIVED, (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、その参加者の配信状態に関係なく、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配信する

マイクやカメラなどのローカルデバイスは、上記の「デバイスからの MediaStream の取得」で説明したのと同じ手順で取得されます。この例では、MediaStream を使用して SDK による配信に使用される 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
    }
}
```

スクリーン共有を配信する

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配信する必要があります。スクリーン共有を配信するには、特にスクリーン共有のメディアを公開するためには、ステージ用の追加のトークンを作成する必要があります。getDisplayMedia を使用し、解像度を最大 720p に制限します。その後、ステップはカメラをステージに配信するのと似ています。

```
// 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) => {
    let 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配信を停止したり、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。

- リモート参加者は配信を停止します。
- 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、StageParticipantInfo で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関する状態情報を参照することもできます。

```
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

`requestQualityStats()` メソッドは、ローカルストリームとリモートストリームの両方に関する詳細な WebRTC 統計情報へのアクセスを提供します。これは LocalStageStream および RemoteStageStream オブジェクトの両方で使用できます。ネットワーク品質、パケット統計、ビットレート情報、フレーム関連のメトリクスなどの包括的な品質メトリクスを返します。

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。ストリームがアクティブでない場合 (内部統計が利用できない、または統計が利用できない場合) は `undefined` が返されます。統計情報が利用可能な場合、ストリーム (リモートまたはローカル、ビデオまたはオーディオ) に応じて、メソッドは LocalVideoStats、LocalAudioStats、RemoteVideoStats、または RemoteAudioStats オブジェクトを返します。

サイマルキャストを含むビデオストリームの場合、配列には複数の統計オブジェクト (レイヤーごとに 1 つ) が含まれていることに留意してください。

ベストプラクティス

- ポーリング頻度 — パフォーマンスへの影響を避けるため、妥当な間隔 (1～5 秒) で `requestQualityStats()` を呼び出します
- エラー処理 — 処理する前に、返された値が `undefined` であるかどうかを必ず確認してください
- メモリ管理 — ストリームが不要になったときに間隔/タイムアウトをクリアします
- ネットワーク品質 — ネットワークによる劣化の可能性に関するユーザーフィードバックには `networkQuality` を使用します 詳細については、「NetworkQuality」を参照してください。

使用例

```
// For local streams
```

```
const localStats = await localVideoStream.requestQualityStats();
const audioStats = await localAudioStream.requestQualityStats();

// For remote streams
const remoteVideoStats = await remoteVideoStream.requestQualityStats();
const remoteAudioStats = await remoteAudioStream.requestQualityStats();

// Example: Monitor stats every 10 seconds
const statsInterval = setInterval(async () => {
   const stats = await localVideoStream.requestQualityStats();
   if (stats) {
      // Note: If simulcast is enabled, you may receive multiple
      // stats records for each layer
      stats.forEach(layer => {
         const rid = layer.rid || 'default';
         console.log(`Layer ${rid}:`, {
            active: layer.active,
            networkQuality: layer.networkQuality,
            packetsSent: layer.packetsSent,
            bytesSent: layer.bytesSent,
            resolution: `${layer.frameWidth}x${layer.frameHeight}`,
            fps: layer.framesPerSecond
         });
      });
   }
}, 10000);
```

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、メディアを再配信する必要があります。

参加者属性を取得

CreateParticipantToken オペレーションリクエストで属性を指定した場合、StageParticipantInfo プロパティに属性が表示されます。

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

補足拡張情報 (SEI)

補足拡張情報 (SEI) NAL ユニットは、フレーム整列メタデータを動画と一緒に保存するために使用されます。H.264 ビデオストリームに配信およびサブスクライブするときに使用できます。SEI ペイロードがサブスクライバーに届くという保証はありません (特にネットワーク状況が悪い場合)。SEI ペイロードは H.264 フレーム構造内にデータを直接保存するため、この機能はオーディオのみのストリームでは利用できません。

SEI ペイロードの挿入

配信元クライアントは、inBandMessaging を有効化するようにビデオの LocalStageStream を設定し、その後に insertSeiMessage メソッド呼び出すことで、配信されているステージストリームに SEI ペイロードを挿入できます。inBandMessaging を有効にすると、SDK のメモリ使用量が増加することに注意してください。

ペイロードは ArrayBuffer タイプである必要があります。ペイロードサイズは 0 KB より大きく 1 KB 未満のサイズにする必要があります。挿入される SEI メッセージの 1 秒あたりの数が 10 KB/秒を超えない必要があります。

```
const config = {
    inBandMessaging: { enabled: true }
};
const vidStream = new LocalStageStream(videoTrack, config);
const payload = new TextEncoder().encode('hello world').buffer;
vidStream.insertSeiMessage(payload);
```

SEI ペイロードの反復

オプションで repeatCount を指定して、送信される次の N 個のフレームで SEI ペイロードの挿入を反復します。これは、ビデオの送信に使用される下層 UDP トランスポートプロトコルが原因で発生する可能性のある特有の損失を軽減するために役立つ場合があります。この値は 0 ~ 30 の値にす

る必要があることに注意してください。受信クライアントには、メッセージを重複除外するロジックが必要です。

```
vidStream.insertSeiMessage(payload, { repeatCount: 5 }); // Optional config,
 repeatCount must be between 0 and 30
```

SEI ペイロードの読み取り

サブスクライブするクライアントは、次の例に示すように、サブスクライバー (複数可) SubscribeConfiguration を設定して inBandMessaging を有効にし、StageEvents.STAGE_STREAM_SEI_MESSAGE_RECEIVED イベントをリッスンすることで、H.264 動画を配信しているパブリッシャー (存在する場合) から SEI ペイロードを読み取ることができます。

```
const strategy = {
    subscribeConfiguration: (participant) => {
        return {
            inBandMessaging: {
                enabled: true
            }
        }
    }
    // ... other strategy functions
}

stage.on(StageEvents.STAGE_STREAM_SEI_MESSAGE_RECEIVED, (participant, seiMessage) => {
    console.log(seiMessage.payload, seiMessage.uuid);
});
```

サイマルキャストによるレイヤードエンコーディング

サイマルキャストによるレイヤードエンコーディングは、パブリッシャーが複数の異なるビデオの品質レイヤーを送信し、サブスクライバーがそれらのレイヤーを動的または手動で変更できるようにする IVS リアルタイムのストリーミング機能です。この機能は、「ストリーミング最適化」ドキュメントで詳しく説明されています。

レイヤードエンコーディングの設定 (パブリッシャー)

パブリッシャーとして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有効にするには、インスタンス化時に LocalStageStream に次の設定を追加します。

```
// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})
```

カメラデバイスの入力解像度に応じて、「ストリーミングの最適化」の「デフォルトレイヤー、品質、フレームレート」セクションで定義されているように、設定された数のレイヤーがエンコードされて送信されます。

また、必要に応じて、サイマルキャスト設定内から個々のレイヤーを設定できます。

```
import { SimulcastLayerPresets } from 'amazon-ivs-web-broadcast'

// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: {
      enabled: true,
      layers: [
         SimulcastLayerPresets.DEFAULT_720,
          SimulcastLayerPresets.DEFAULT_360,
          SimulcastLayerPresets.DEFAULT_180,
   }
})
```

または、最大で 3 つのレイヤー用に独自のカスタムレイヤー設定を作成することもできます。空のアレイを指定するか、値を指定しない場合、上記のデフォルトが使用されます。レイヤーは、次の必須プロパティで説明されています。

- `height: number;`
- `width: number;`
- `maxBitrateKbps: number;`
- `maxFramerate: number;`

プリセットから、個々のプロパティを上書きするか、まったく新しい設定を作成できます。

```
import { SimulcastLayerPresets } from 'amazon-ivs-web-broadcast'

const custom720pLayer = {
   ...SimulcastLayerPresets.DEFAULT_720,
```

```
   maxFramerate: 15,
}

const custom360pLayer = {
       maxBitrateKbps: 600,
       maxFramerate: 15,
       width: 640,
       height: 360,
}

// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: {
      enabled: true,
      layers: [
         custom720pLayer,
         custom360pLayer,
   }
})
```

個々のレイヤーを設定するときにトリガーできる最大値、制限、エラーについては、SDK リファレンスドキュメントを参照してください。

レイヤードエンコーディングの設定 (サブスクライバー)

サブスクライバーとして、レイヤードエンコーディングを有効にするために必要なものはありません。パブリッシャーがサイマルキャストレイヤーを送信している場合、デフォルトでサーバーによってレイヤー間で動的に適応され、サブスクライバーのデバイスおよびネットワークの状態に基づいて最適な品質が選択されます。

あるいは、パブリッシャーが送信している明示的なレイヤーを選択するには、以下に説明するいくつかのオプションがあります。

オプション 1: 初期レイヤー品質の選択

subscribeConfiguration 戦略を使用すると、サブスクライバーとして受信する初期レイヤーを選択できます。

```
const strategy = {
    subscribeConfiguration: (participant) => {
        return {
            simulcast: {
```

```
                initialLayerPreference: InitialLayerPreference.LOWEST_QUALITY
            }
        }
    }
    // ... other strategy functions
}
```

デフォルトでは、サブスクライバーは常に最初に最低品質のレイヤーが送信されます。これにより、徐々に最高品質のレイヤーにまで拡大します。エンドユーザーの帯域幅の消費量が最適化され、ビデオ再生に最適な時間が実現されるため、より貧弱なネットワーク上のユーザーに対して初期ビデオフリーズが軽減されます。

これらのオプションは InitialLayerPreference で利用できます。

- LOWEST_QUALITY — サーバーは、最初に最低品質のビデオレイヤーを配信します。帯域幅の消費とメディアの時間が最適化されます。品質はビデオのサイズ、ビットレート、フレームレートの組み合わせとして定義されます。例えば、720p ビデオは 1080p ビデオよりも品質が低くなります。
- HIGHEST_QUALITY — サーバーは、最初に最高品質のビデオレイヤーを配信します。品質が最適化されますが、メディアの時間が長くなる場合があります。品質はビデオのサイズ、ビットレート、フレームレートの組み合わせとして定義されます。例えば、1080p ビデオは 720p ビデオよりも高品質です。

注: 初期レイヤー設定 (initialLayerPreference の呼び出し) を反映させるには、再サブスクライブが必要です。これらの更新はアクティブなサブスクリプションに適用されないためです。

オプション 2: ストリームに優先されるレイヤー

ストリームが開始されたら、preferredLayerForStream 戦略メソッドを使用できます。この戦略メソッドは、参加者およびストリーム情報を公開します。

戦略メソッドは、次の内容として返すことができます。

- RemoteStageStream.getLayers が返す内容に直接的に基づくレイヤーオブジェクト
- StageStreamLayer.label に基づく、レイヤーオブジェクトのラベル文字列
- レイヤーを選択せず、動的適応が優先されることを示す未定義または null

例えば、次の戦略ではユーザーが常に最低品質のビデオレイヤーを選択するようにします。

```
const strategy = {
    preferredLayerForStream: (participant, stream) => {
        return stream.getLowestQualityLayer();
    }
    // ... other strategy functions
}
```

レイヤーの選択をリセットして動的適応に戻るには、戦略で null または未定義を返します。この例では、appState は可能なアプリケーションの状態を表すダミー変数です。

```
const strategy = {
    preferredLayerForStream: (participant, stream) => {
        if (appState.isAutoMode) {
            return null;
        } else {
            return appState.layerChoice
        }
    }
    // ... other strategy functions
}
```

オプション 3: RemoteStageStream レイヤーヘルパー

RemoteStageStream には、レイヤーの選択について決定し、対応する選択をエンドユーザーに表示するために使用できるいくつかのヘルパーがあります。

- レイヤーイベント - StageEvents に加え、RemoteStageStream オブジェクト自体にはレイヤーおよびサイマルキャストの適応変更を伝えるイベントがあります。
  - stream.on(RemoteStageStreamEvents.ADAPTION_CHANGED, (isAdapting) => {})
  - stream.on(RemoteStageStreamEvents.LAYERS_CHANGED, (layers) => {})
  - stream.on(RemoteStageStreamEvents.LAYER_SELECTED, (layer, reason) => {})
- レイヤーメソッド — RemoteStageStream には、ストリームおよび提示されるレイヤーに関する情報を取得するために使用できるいくつかのヘルパーメソッドがあります。これらのメソッドは、preferredLayerForStream 戦略で提供されるリモートストリームに加え、StageEvents.STAGE_PARTICIPANT_STREAMS_ADDED を介して配信されるリモートストリームで利用できます。
  - stream.getLayers

- stream.getSelectedLayer
- stream.getLowestQualityLayer
- stream.getHighestQualityLayer

詳細については、「SDK リファレンスドキュメント」の「RemoteStageStream」クラスを参照してください。LAYER_SELECTED の理由として UNAVAILABLE が返された場合、これはリクエストされたレイヤーが選択できなかったことを示します。代わりにベストエフォートの選択が行われ、ストリームの安定性を維持するために、通常は低品質のレイヤーが選択されます。

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
         return;
      case StageConnectionState.CONNECTING:
         // handle establishing connection UI
         return;
      case StageConnectionState.CONNECTED:
         // SDK is connected to the Stage
         return;
      case StageConnectionState.ERRORED:
         // SDK encountered an error and lost its connection to the stage. Wait for
 CONNECTED.
         return;
    }
})
```

通常、SDK は内部的に復旧しようとするため、ステージに正常に参加した後に発生するエラー状態を無視できます。SDK が ERRORED 状態を報告し、ステージが長時間 (30 秒以上) CONNECTING 状態のままである場合、おそらくネットワークから切断されています。

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。

IVS Web Broadcast SDK の既知の問題と回避策 | Real-Time Streaming

このドキュメントでは、Amazon IVS Real-Time Streaming Web Broadcast SDK を使用する際に発生する可能性のある既知の問題の一覧を示し、可能な回避策を提案します。

- `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可能性があり、`StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` イベントを聞くと`StageParticipantPublishState.ATTEMPTING_PUBLISH` 状態が繰り返される可能性があります。

  回避策: `getUserMedia` または `getDisplayMedia` を呼び出すときは、ビデオ解像度を 720p に制限してください。具体的には、幅と高さの `getUserMedia` と `getDisplayMedia` の制約値は、乗算時に 921600 (1280 x 720) を超えることはできません。

- `stage.leave()` が呼び出されるか、リモート参加者が退出すると、ブラウザのデバッグコンソールに 404 DELETE エラーが表示されます。

  回避策: 該当なし。これは無害なエラーです。

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。アクセス許可を有効にした後、正常に動作するために 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、ブラウザで NotFoundError 例外がスローされます。
- `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 の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配信が停止します。

  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。

- ブラウザのバックグラウンド設定を行うと、配信ストリームが真っ暗になり、音声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からです。

IVS Web Broadcast SDK でのエラー処理 | Real-Time Streaming

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る処理について説明します。エラーは SDK によって `StageEvents.ERROR` イベントのリスナーに報告されます。

```
stage.on(StageEvents.ERROR, (error: StageError) => {
    // log or handle errors here
    console.log(`${error.code}, ${error.category}, ${error.message}`);
});
```

ステージエラー

StageError は、SDK が復旧できない問題に遭遇したときに報告され、通常、アプリケーションの介入やネットワークの再接続が必要になります。

報告されたそれぞれの `StageError` には、コード (または `StageErrorCode`)、メッセージ (文字列)、カテゴリ (`StageErrorCategory`) があります。それぞれが基盤となるオペレーションカテゴリに関連しています。

エラーのオペレーションカテゴリは、ステージへの接続 (`JOIN_ERROR`)、ステージへのメディアの送信 (`PUBLISH_ERROR`)、またはステージからの受信メディアストリームの受信 (`SUBSCRIBE_ERROR`) のどちらに関連しているかに基づいて決定されます。

`StageError` のコードプロパティは、特定の問題をレポートします。

名前	コード	[Recommended Action] (推奨されるアクション)
TOKEN_MALFORMED	1	有効なトークンを作成し、ステージのインスタンス化を再試行してください。
TOKEN_EXPIRED	2	有効期限が切れていないトークンを作成し、ステージのインスタンス化を再試行します。
タイムアウト	3	オペレーションがタイムアウトしました。ステージが存在し、トークンが有効である場合、この障害はネットワークの問題である可能性があります。この場合は、デバイスの接続が回復するまでお待ちください。
FAILED	4	オペレーションの試行中に致命的な状態が発生しました。エラーの詳細を確認してください。 ステージが存在し、トークンが有効である場合、この障害はネットワークの問題である可能性があります。この場合は、デバイスの接続が回復するまでお待ちください。 ネットワークの安定性に関連するほとんどの障害の場合、SDK は FAILED エラーを出力する前に最大 30 秒間内部で再試行します。
CANCELED	5	アプリケーションコードをチェックし、繰り返し `join`、`refreshStrategy` 、または `replaceStrategy` 呼び出しがないことを確認します。これにより、完了前に繰り返しオペレーションが開始され、キャンセルされる可能性があります。
STAGE_AT_CAPACITY	6	このエラーは、ステージまたはアカウントのキャパシティに達していることを示します。ステージが参加者の制限に達した場合は、戦略を更新して、ステージのキャパシティが解放されたときにオペレーションを再試行してください。アカウントが同時サブスクリプションまたは同時パブリッシャーのクォータに達した場合は、AWS Service Quotas コンソールを使用して

名前	コード	[Recommended Action] (推奨されるアクション)
		使用量を減らすか、クォータの引き上げをリクエストしてください。
CODEC_MISMATCH	7	コーデックはステージではサポートされていません。コーデックのサポートについては、ブラウザとプラットフォームを確認してください。IVS Real-Time Streaming の場合、ブラウザはビデオ用の H.264 コーデックとオーディオ用の Opus コーデックをサポートしている必要があります。
TOKEN_NOT_ALLOWED	8	トークンにはオペレーションに対するアクセス許可がありません。トークンを正しいアクセス許可で再作成し、再試行してください。
STAGE_DELETED	9	なし。削除されたステージに参加しようとすると、このエラーがトリガーされます。
PARTICIPANT_DISCONNECTED	10	なし。切断された参加者のトークンを使用して参加しようとすると、このエラーがトリガーされます。

StageError の処理の例

StageError コードを使用して、エラーが期限切れのトークンによるものかどうかを判断します。

```
stage.on(StageEvents.ERROR, (error: StageError) => {
    if (error.code === StageError.TOKEN_EXPIRED) {
        // recreate the token and stage instance and re-join
    }
});
```

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。

配信中/サブスクライブ中の場合は、配信/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。

エラー状態

これらの状態は、アプリケーションのログ記録に使用し、特定の参加者のステージへの接続の問題を通知するメッセージをユーザーに表示することをお勧めします。

配信

SDK は配信が失敗したときに ERRORED を報告します。

```
stage.on(StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED, (participantInfo, state)
 => {
  if (state === StageParticipantPublishState.ERRORED) {
      // Log and/or display message to user
  }
});
```

Subscribe

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) {
    // Log and/or display message to user
  }
});
```

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

IVS Real-Time Streaming 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配信する
- 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
- ステージに配信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: 1.40.0 (リリースノート)

リファレンスドキュメント: Amazon IVS Android プレイヤーで利用可能な最も重要な方法については、 https://aws.github.io/amazon-ivs-broadcast-docs/1.40.0/android/ にある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: GitHub の Android サンプルリポジトリ (https://github.com/aws-samples/amazon-ivs-real-time-streaming-android-samples) を参照してください。

プラットフォーム要件: Android 9.0 以降

IVS Android Broadcast SDK の開始方法 | Real-Time Streaming

このドキュメントでは、IVS Real-Time Streaming Android Broadcast SDK の使用を開始するためのステップについて説明します。

ライブラリのインストール

Amazon IVS Android ブロードキャストライブラリを Android 開発環境に追加する方法は以下のようにいくつかあります。Gradle を直接使用する、Gradle バージョンカタログを使用する、または SDK を手動でインストールする。

Gradle を直接使用する: 次のように、ライブラリをモジュールの `build.gradle` ファイルに追加します (IVS Broadcast SDK の最新バージョンの場合)。

```
repositories {
    mavenCentral()
}

dependencies {
```

```
    implementation 'com.amazonaws:ivs-broadcast:1.40.0:stages@aar'
}
```

Gradle バージョンカタログを使用する: まず、以下をモジュールの build.gradle ファイルに含めます。

```
implementation(libs.ivs){
   artifact {
      classifier = "stages"
      type = "aar"
   }
}
```

次に、libs.version.toml ファイルに以下を含めます (IVS Broadcast SDK の最新バージョンの場合)。

```
[versions]
ivs="1.40.0"

[libraries]
ivs = {module = "com.amazonaws:ivs-broadcast", version.ref = "ivs"}
```

SDK を手動でインストールする: 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 -stages が付いた aar をダウンロードしてください。

また、スピーカーフォンに対する SDK コントロールも許可します。どのインストール方法を選択しても、次のアクセス許可をマニフェストに追加して、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"/>
```

デバッグ情報を含む SDK の使用

また、デバッグ情報を含む Android Broadcast SDK のバージョンも公開しています。このバージョンを使用すると、IVS Broadcast SDK (つまり、libbroadcastcore.so) でクラッシュした場合、Firebase Crashlytics のデバッグレポート (スタックトレース) の品質を向上させることができま

す。これらのクラッシュを IVS SDK チームに報告すると、高品質のスタックトレースにより、問題の修正が容易になります。

このバージョンの SDK を使用するには、Gradle ビルドファイルに以下を含めます。

```
implementation "com.amazonaws:ivs-broadcast:$version:stages-unstripped@aar"
```

以下の代わりに上記の行を使用します。

```
implementation "com.amazonaws:ivs-broadcast:$version:stages@aar"
```

Firebase Crashlytics へのシンボルのアップロード

Gradle ビルドファイルが Firebase Crashlytics 用に設定されていることを確認します。以下の Google の指示に従ってください。

https://firebase.google.com/docs/crashlytics/ndk-reports

`com.google.firebase:firebase-crashlytics-ndk` を依存関係として必ず含めてください。

リリース用にアプリを構築する場合、Firebase Crashlytics プラグインは情報を自動的にアップロードするはずです。情報を手動でアップロードするには、次のいずれかを実行します。

```
gradle uploadCrashlyticsSymbolFileRelease
```

```
./gradlew uploadCrashlyticsSymbolFileRelease
```

(情報が自動と手動の両方で 2 回アップロードされても問題ありません)。

Release .apk の肥大化を防ぐ

リリース `.apk` ファイルをパッケージ化する前に、Android Gradle Plugin は共有ライブラリ (IVS Broadcast SDK の `libbroadcastcore.so` ライブラリを含む) からデバッグ情報を自動的に削除しようとします。ただし、これはときに動作しないことがあります。その結果、`.apk` ファイルが肥大化し、Android Gradle プラグインから、「デバッグ情報を削除できず、そのまま `.so` ファイルをパッケージ化している」という警告メッセージが表示される可能性があります。このような場合は、以下のことを試してみます。

- Android NDK をインストールします。すべての最新バージョンが機能します。

- ndkVersion <your_installed_ndk_version_number> をアプリケーションの build.gradle ファイルに追加します。アプリケーション自体にネイティブコードが含まれていない場合でも、これを行います。

詳細については、「問題レポート」を参照してください。

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();
    }
}
```

IVS Android Broadcast SDK での配信とサブスクライブ | Real-Time Streaming

このドキュメントでは、IVS Real-Time Streaming Android Broadcast SDK を使用してステージに配信とサブスクライブを行うためのステップについて説明します。

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配信の状態、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配信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。

参加者へのサブスクライブの設定

```
SubscribeConfiguration subscribeConfigurationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo);
```

リモート参加者がサブスクライブしている場合 (「参加者へのサブスクライブ」を参照)、SDK はホストアプリケーションにその参加者のカスタムサブスクライブ設定についてクエリします。この設定はオプションであり、ホストアプリケーションがサブスクライバーの動作の特定の側面を制御できるようにします。設定できる内容の詳細については、SDK リファレンスドキュメントの「SubscribeConfiguration」を参照してください。

次に示すのは実装の例です。

```
@Override
public SubscribeConfiguration subscribeConfigrationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo) {
    SubscribeConfiguration config = new SubscribeConfiguration();


 config.jitterBuffer.setMinDelay(JitterBufferConfiguration.JitterBufferDelay.MEDIUM());

    return config;
}
```

この実装では、サブスクライブしたすべての参加者のジッターバッファ最小遅延を MEDIUM のプリセットに更新します。

shouldSubscribeToParticipant を使用した、より高度な実装も可能です。指定された ParticipantInfo を使用して、特定の参加者のサブスクライブ設定を選択的に更新できます。

デフォルトの動作を使用することをお勧めします。カスタム設定は、特定の動作を変更したい場合にのみ指定します。

配信

```
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、特定の参加者を配信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配信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、ユーザーは常に配信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。(配信/配信停止も使用できますが、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。)

配信するストリームの選択

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

配信時には、これを使用して配信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配信」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、エンドユーザーがボタンをタップするまでホストアプリケーションが配信したくない場合、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、配信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);

void onStreamAdaptionChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull RemoteStageStream stream, boolean adaption);
```

```
void onStreamLayersChanged(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull RemoteStageStream stream, @NonNull
 List<RemoteStageStream.Layer> layers);

void onStreamLayerSelected(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo, @NonNull RemoteStageStream stream, @Nullable RemoteStageStream.Layer
 layer, @NonNull RemoteStageStream.LayerSelectedReason reason);
```

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、その参加者の配信状態に関係なく、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
```

配信参加者のみが onParticipantJoined をトリガーし、参加者が配信を停止するか、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配信する

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 が配信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
```

```
});
```

参加者が配信を停止するか、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。

- リモート参加者は配信を停止します。
- 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

配信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 オペレーションリクエストで属性を指定した場合、ParticipantInfo プロパティに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());
 }
}
```

メッセージを埋め込む

ImageDevice で embedMessage メソッドを使用すると、配信中のビデオフレームにメタデータペイロードを直接挿入できます。これは、リアルタイムアプリケーションのフレーム同期型メッセージン

グを可能にします。メッセージの埋め込みを使用できるのは、リアルタイム配信(低レイテンシー配信ではない) の SDK を使用している場合のみです。

埋め込みメッセージは、ビデオフレーム内に直接埋め込まれ、パケット配信を保証しない UDP 経由で送信されるため、必ずしもサブスクライバーに届くとは限りません。送信中のパケット損失は、特にネットワーク状態が悪い場合において、メッセージの損失につながる可能性があります。この問題を軽減するため、embedMessage メソッドには repeatCount パラメータが含まれています。このパラメータは、連続する複数のフレーム全体でメッセージを複製することで、配信信頼性を向上させます。この機能を利用できるのはビデオストリームのみです。

embedMessage の使用

配信元のクライアントは、ImageDevice で embedMessage メソッドを使用することでメッセージペイロードをビデオストリームに埋め込むことができます。ペイロードサイズは 0 KB より大きく 1 KB 未満のサイズにする必要があります。挿入される埋め込みメッセージの 1 秒あたりの数が 10 KB/秒を超えないようにする必要があります。

```
val surfaceSource: SurfaceSource = imageStream.device as SurfaceSource
val message = "hello world"
val messageBytes = message.toByteArray(StandardCharsets.UTF_8)

try {
    surfaceSource.embedMessage(messageBytes, 0)
} catch (e: BroadcastException) {
    Log.e("EmbedMessage", "Failed to embed message: ${e.message}")
}
```

メッセージペイロードの反復

repeatCount を使用して複数のフレーム全体でメッセージを複製し、信頼性を向上させます。この値は 0 ~ 30 の範囲の値にする必要があります。受信クライアントには、メッセージを重複除外するロジックが必要です。

```
try {
    surfaceSource.embedMessage(messageBytes, 5)
    // repeatCount: 0-30, receiving clients should handle duplicates
} catch (e: BroadcastException) {
    Log.e("EmbedMessage", "Failed to embed message: ${e.message}")
}
```

埋め込みメッセージの読み取り

受信ストリームから埋め込みメッセージを読み取る方法については、以下の「補足拡張情報 (SEI) を取得する」を参照してください。

補足拡張情報 (SEI、Supplemental Enhancement Information) を取得する

補足拡張情報 (SEI) NAL ユニットは、フレーム整列メタデータを動画と一緒に保存するために使用されます。パブリッシャーの ImageDevice から送信される ImageDeviceFrame オブジェクトの embeddedMessages プロパティを調べることにより、サブスクライブしているクライアントは H.264 ビデオを配信しているパブリッシャーから SEI ペイロードを読み取ることができます。これを行うには、次の例で示すように、パブリッシャーの ImageDevice を取得し、setOnFrameCallback に提供されるコールバックを介して各フレームを確認します。

```
// in a StageRenderer’s onStreamsAdded function, after acquiring the new ImageStream

val imageDevice = imageStream.device as ImageDevice
imageDevice.setOnFrameCallback(object : ImageDevice.FrameCallback {
 override fun onFrame(frame: ImageDeviceFrame) {
      for (message in frame.embeddedMessages) {
          if (message is UserDataUnregisteredSeiMessage) {
              val seiMessageBytes = message.data
              val seiMessageUUID = message.uuid

              // interpret the message's data based on the UUID
          }
      }
 }
})
```

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、配信を停止するか、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 実装を更新して配信を停止し、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

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、パブリッシャーが複数の異なるビデオの品質レイヤーを送信し、サブスクライバーがそれらのレイヤーを動的または手動で設定できるようにする IVS リアルタイムのストリーミング機能です。この機能は、「ストリーミング最適化」ドキュメントで詳しく説明されています。

レイヤードエンコーディングの設定 (パブリッシャー)

パブリッシャーとして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有効にするには、インスタンス化時に LocalStageStream に次の設定を追加します。

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

ビデオ設定で設定した解像度に応じて、「ストリーミングの最適化」の「デフォルトレイヤー、品質、フレームレート」セクションで定義されているように、設定された数のレイヤーがエンコードされて送信されます。

また、必要に応じて、サイマルキャスト設定内から個々のレイヤーを設定できます。

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

List<StageVideoConfiguration.Simulcast.Layer> simulcastLayers = new ArrayList<>();
simulcastLayers.add(StagePresets.SimulcastLocalLayer.DEFAULT_720);
simulcastLayers.add(StagePresets.SimulcastLocalLayer.DEFAULT_180);

config.simulcast.setLayers(simulcastLayers);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

または、最大で 3 つのレイヤー用に独自のカスタムレイヤー設定を作成することもできます。空のアレイを指定するか、値を指定しない場合、上記のデフォルトが使用されます。レイヤーは、次の必須プロパティセッターで説明されています。

- `setSize: Vec2;`
- `setMaxBitrate: integer;`
- `setMinBitrate: integer;`
- `setTargetFramerate: integer;`

プリセットから、個々のプロパティを上書きするか、まったく新しい設定を作成できます。

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

List<StageVideoConfiguration.Simulcast.Layer> simulcastLayers = new ArrayList<>();

// Configure high quality layer with custom framerate
StageVideoConfiguration.Simulcast.Layer customHiLayer =
 StagePresets.SimulcastLocalLayer.DEFAULT_720;
```

```
customHiLayer.setTargetFramerate(15);

// Add layers to the list
simulcastLayers.add(customHiLayer);
simulcastLayers.add(StagePresets.SimulcastLocalLayer.DEFAULT_180);

config.simulcast.setLayers(simulcastLayers);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

個々のレイヤーを設定するときにトリガーできる最大値、制限、エラーについては、SDK リファレンスドキュメントを参照してください。

レイヤードエンコーディングの設定 (サブスクライバー)

サブスクライバーとして、レイヤードエンコーディングを有効にするために必要なものはありません。パブリッシャーがサイマルキャストレイヤーを送信している場合、デフォルトでサーバーによってレイヤー間で動的に適応され、サブスクライバーのデバイスおよびネットワークの状態に基づいて最適な品質が選択されます。

あるいは、パブリッシャーが送信している明示的なレイヤーを選択するには、以下に説明するいくつかのオプションがあります。

オプション 1: 初期レイヤー品質の選択

subscribeConfigurationForParticipant 戦略を使用すると、サブスクライバーとして受信する初期レイヤーを選択できます。

```
@Override
public SubscribeConfiguration subscribeConfigrationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo) {
    SubscribeConfiguration config = new SubscribeConfiguration();


 config.simulcast.setInitialLayerPreference(SubscribeSimulcastConfiguration.InitialLayerPrefere

    return config;
}
```

デフォルトでは、サブスクライバーは常に最初に最低品質のレイヤーが送信されます。これにより、徐々に最高品質のレイヤーにまで拡大します。エンドユーザーの帯域幅の消費量が最適化され、ビデオ再生に最適な時間が実現されるため、より貧弱なネットワーク上のユーザーに対して初期ビデオフリーズが軽減されます。

これらのオプションは InitialLayerPreference で利用できます。

- LOWEST_QUALITY — サーバーは、最初に最低品質のビデオレイヤーを配信します。帯域幅の消費とメディアの時間が最適化されます。品質はビデオのサイズ、ビットレート、フレームレートの組み合わせとして定義されます。例えば、720p ビデオは 1080p ビデオよりも品質が低くなります。
- HIGHEST_QUALITY — サーバーは、最初に最高品質のビデオレイヤーを配信します。品質が最適化されますが、メディアの時間が長くなる場合があります。品質はビデオのサイズ、ビットレート、フレームレートの組み合わせとして定義されます。例えば、1080p ビデオは 720p ビデオよりも高品質です。

注: 初期レイヤー設定 (setInitialLayerPreference の呼び出し) を反映させるには、再サブスクライブが必要です。これらの更新はアクティブなサブスクリプションに適用されないためです。

オプション 2: ストリームに優先されるレイヤー

preferredLayerForStream 戦略メソッドを使用すると、ストリームの開始後にレイヤーを選択できます。この戦略メソッドは参加者とストリーム情報を受け取るため、参加者ごとにレイヤーを選択できます。このメソッドは、ストリームのレイヤーが変化したとき、参加者の状態が変化したとき、またはホストアプリケーションが戦略を更新したときなど、特定のイベントに応じて SDK が呼び出します。

この戦略メソッドは RemoteStageStream.Layer オブジェクトを返します。これは次のいずれかになります。

- RemoteStageStream.getLayers によって返されるレイヤーオブジェクトなど。
- レイヤーを選択せず、動的適応が優先されることを示す null。

例えば、次の戦略ではユーザーが常に最低品質のビデオレイヤーを選択するようにします。

```
@Nullable
@Override
public RemoteStageStream.Layer preferredLayerForStream(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo, @NonNull RemoteStageStream stream) {
    return stream.getLowestQualityLayer();
```

```
}
```

レイヤーの選択をリセットして動的適応に戻るには、戦略で null または未定義を返します。この例では、appState はホストアプリケーションの状態を表すプレースホルダー変数です。

```
@Nullable
@Override
public RemoteStageStream.Layer preferredLayerForStream(@NonNull Stage stage, @NonNull
 ParticipantInfo participantInfo, @NonNull RemoteStageStream stream) {
    if (appState.isAutoMode) {
        return null;
    } else {
        return appState.layerChoice;
    }
}
```

オプション 3: RemoteStageStream レイヤーヘルパー

RemoteStageStream には、レイヤーの選択について決定し、対応する選択をエンドユーザーに表示するために使用できるいくつかのヘルパーがあります。

- レイヤーイベント — StageRenderer に加え、RemoteStageStream.Listener にはレイヤーおよびサイマルキャストの適応変更を伝えるイベントがあります。
  - `void onAdaptionChanged(boolean adaption)`
  - `void onLayersChanged(@NonNull List<Layer> layers)`
  - `void onLayerSelected(@Nullable Layer layer, @NonNull LayerSelectedReason reason)`
- レイヤーメソッド — RemoteStageStream には、ストリームおよび提示されるレイヤーに関する情報を取得するために使用できるいくつかのヘルパーメソッドがあります。これらのメソッドは、preferredLayerForStream 戦略で提供されるリモートストリームに加え、StageRenderer.onStreamsAdded を介して配信されるリモートストリームで利用できます。
  - `stream.getLayers`
  - `stream.getSelectedLayer`
  - `stream.getLowestQualityLayer`
  - `stream.getHighestQualityLayer`
  - `stream.getLayersWithConstraints`

詳細については、「SDK リファレンスドキュメント」の「RemoteStageStream」クラスを参照してください。LayerSelected の理由として UNAVAILABLE が返された場合、これはリクエストされたレイヤーが選択できなかったことを示します。代わりにベスト工フォートの選択が行われ、ストリームの安定性を維持するために、通常は低品質のレイヤーが選択されます。

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:「不明なネットワークエラーにより PeerConnection が失われました」
- 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配信するには、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);
```

IVS Android Broadcast SDK の既知の問題と回避策 | Real-Time Streaming

このドキュメントでは、Amazon IVS Real-Time Streaming Android Broadcast SDK を使用する際に発生する可能性のある既知の問題の一覧を示し、可能な回避策を提案します。

- 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工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配信中でも、サブスクライバーが受け取る配信状態が `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、音声を配信停止にしてから再配信して問題を解決できます。
- ステージに配信する場合、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配信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配信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、176x176 未満のビデオサイズで設定することはできません。サイズを小さく設定するとエラーが発生し、ストリーミングが停止します。

  回避策： ビデオサイズを 176x176 未満に設定しないでください。

IVS Android Broadcast SDK でのエラー処理 | Real-Time Streaming

このセクションでは、エラー状態の概要、IVS Real-Time Streaming Android 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る処理について説明します。

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、配信/サブスクライブ状態に関連しており、配信/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。

配信/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。再試行中の配信/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

配信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、配信/サブスクライブ操作を再試行します。再試行中の配信/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 Real-Time Streaming 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配信する
- 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
- ステージに配信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: 1.40.0 (リリースノート)

リファレンスドキュメント: Amazon IVS iOS プレイヤーで利用できる最も重要な方法については、https://aws.github.io/amazon-ivs-broadcast-docs/1.40.0/ios/ にある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: GitHub の iOS サンプルリポジトリ (https://github.com/aws-samples/amazon-ivs-real-time-streaming-ios-samples) を参照してください。

プラットフォームの要件: iOS 14 以降

IVS iOS Broadcast SDK の開始方法 | Real-Time Streaming

このドキュメントでは、IVS Real-Time Streaming iOS Broadcast SDK の使用を開始するためのステップについて説明します。

ライブラリのインストール

Swift Package Manager を介して Broadcast SDK を統合することをお勧めします。(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 (Swift Package Manager)

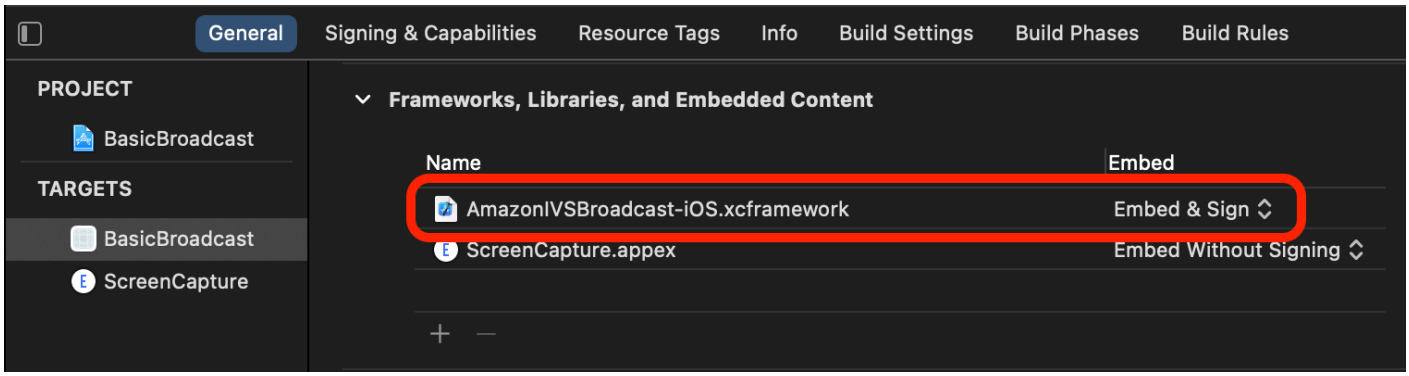
1. Package.swift ファイルを「https://broadcast.live-video.net/1.40.0/Package.swift」からダウンロードしてください。
2. プロジェクトで AmazonIVSBroadcast という名前の新しいディレクトリを作成し、バージョン管理に追加します。
3. ダウンロードした Package.swift ファイルを新しいディレクトリに配置します。
4. Xcode で [ファイル] > [パッケージの依存関係を追加] に移動し、[ローカルに追加] を選択します。
5. 作成した AmazonIVSBroadcast ディレクトリに移動して選択たら、パッケージの追加を選択します。

6. AmazonIVSBroadcast のパッケージ製品の選択を求められたら、「ターゲットに追加」セクションでアプリケーションターゲットを設定し、パッケージ製品として [AmazonIVSBroadcastStages] を選択します。
7. パッケージの追加を選択します。

重要: IVS リアルタイムストリーミングの Broadcast SDK 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。

代替方法: フレームワークを手動でインストールする

1. 次のリンクから最新バージョンをダウンロードします。 https://broadcast.live-video.net/1.40.0/AmazonIVSBroadcast-Stages.xcframework.zip.
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
}
```

IVS iOS Broadcast SDK での配信とサブスクライブ | Real-Time Streaming

このドキュメントでは、IVS Real-Time Streaming iOS Broadcast SDK を使用してステージに配信とサブスクライブを行うためのステップについて説明します。

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配信の状態、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配信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。

参加者へのサブスクライブの設定

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration
```

リモート参加者がサブスクライブしている場合 (「参加者へのサブスクライブ」を参照)、SDK はホストアプリケーションにその参加者のカスタムサブスクライブ設定についてクエリします。この設定はオプションであり、ホストアプリケーションがサブスクライバーの動作の特定の側面を制御できるようにします。設定できる内容の詳細については、SDK リファレンスドキュメントの「SubscribeConfiguration」を参照してください。

次に示すのは実装の例です。

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration {
    let config = IVSSubscribeConfiguration()
```

```
    try! config.jitterBuffer.setMinDelay(.medium())

    return config
}
```

この実装では、サブスクライブしたすべての参加者のジッターバッファ最小遅延を MEDIUM のプリセットに更新します。

shouldSubscribeToParticipant を使用した、より高度な実装も可能です。指定された ParticipantInfo を使用して、特定の参加者のサブスクライブ設定を選択的に更新できます。

デフォルトの動作を使用することをお勧めします。カスタム設定は、特定の動作を変更したい場合にのみ指定します。

配信

```
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、特定の参加者を配信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配信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、ユーザーは常に配信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。(配信/配信停止も使用できますが、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。)

配信するストリームの選択

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

配信時には、これを使用して配信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配信」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、エンドユーザーがボタンをタップするまでホストアプリケーションが配信したくない場合、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、配信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?)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, stream:
 IVSRemoteStageStream, didChangeStreamAdaption adaption: Bool)

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, stream:
 IVSRemoteStageStream, didChange layers: [IVSRemoteStageStreamLayer])

func stage(_ stage: IVSStage, participant: IVSParticipantInfo, stream:
 IVSRemoteStageStream, didSelect layer: IVSRemoteStageStreamLayer?, reason:
 IVSRemoteStageStream.LayerSelectedReason)
```

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、その参加者の配信状態に関係なく、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。

配信参加者のみが participantDidJoin をトリガーし、参加者が配信を停止するか、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配信する

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 で配信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 and create a stream
let microphone = devices.compactMap({ $0 as? IVSMicrophone }).first!
let microphoneStream = IVSLocalStageStream(device: microphone)

// Configure the audio manager to use the videoChat preset, which is optimized for bi-
directional communication, including echo cancellation.
IVSStageAudioManager.sharedInstance().setPreset(.videoChat)

// 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 */
    })
}
```

参加者が配信を停止するか、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。

- リモート参加者は配信を停止します。

- 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

配信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 オペレーションリクエストで属性を指定した場合、IVSParticipantInfo プロパティに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)")
    }
}
```

メッセージを埋め込む

IVSImageDevice で embedMessage メソッドを使用すると、配信中のビデオフレームにメタデータペイロードを直接挿入できます。これは、リアルタイムアプリケーションのフレーム同期型メッセージングを可能にします。メッセージの埋め込みを使用できるのは、リアルタイム配信(低レイテンシー配信ではない) の SDK を使用している場合のみです。

埋め込みメッセージは、ビデオフレーム内に直接埋め込まれ、パケット配信を保証しない UDP 経由で送信されるため、必ずしもサブスクライバーに届くとは限りません。送信中のパケット損失は、特

にネットワーク状態が悪い場合において、メッセージの損失につながる可能性があります。この問題を軽減するため、embedMessage メソッドには repeatCount パラメータが含まれています。このパラメータは、連続する複数のフレーム全体でメッセージを複製することで、配信信頼性を向上させます。この機能を利用できるのはビデオストリームのみです。

embedMessage の使用

配信元のクライアントは、IVSImageDevice で embedMessage メソッドを使用して、メッセージペイロードをビデオストリームに埋め込むことができます。ペイロードサイズは 0 KB より大きく 1 KB 未満のサイズにする必要があります。挿入される埋め込みメッセージの 1 秒あたりの数が 10 KB/秒を超えないようにする必要があります。

```
let imageDevice: IVSImageDevice = imageStream.device as! IVSImageDevice
let messageData = Data("hello world".utf8)

do {
    try imageDevice.embedMessage(messageData, withRepeatCount: 0)
} catch {
    print("Failed to embed message: \(error)")
}
```

メッセージペイロードの反復

repeatCount を使用して複数のフレーム全体でメッセージを複製し、信頼性を向上させます。この値は 0 ~ 30 の範囲の値にする必要があります。受信クライアントには、メッセージを重複除外するロジックが必要です。

```
try imageDevice.embedMessage(messageData, withRepeatCount: 5)

// repeatCount: 0-30, receiving clients should handle duplicates
```

埋め込みメッセージの読み取り

受信ストリームから埋め込みメッセージを読み取る方法については、以下の「補足拡張情報 (SEI) を取得する」を参照してください。

補足拡張情報 (SEI、Supplemental Enhancement Information) を取得する

補足拡張情報 (SEI) NAL ユニットは、フレーム整列メタデータを動画と一緒に保存するために使用されます。パブリッシャーの IVSImageDevice から送信される IVSImageDeviceFrame オブジェクトの embeddedMessages プロパティを調べることにより、サブスクライブしているク

ライアントは H.264 ビデオを配信しているパブリッシャーから SEI ペイロードを読み取ることができます。これを行うには、次の例で示すように、パブリッシャーの IVSImageDevice を取得し、setOnFrameCallback に提供されるコールバックを介して各フレームを確認します。

```
// in an IVSStageRenderer’s stage:participant:didAddStreams: function, after acquiring
 the new IVSImageStream

let imageDevice: IVSImageDevice? = imageStream.device as? IVSImageDevice
imageDevice?.setOnFrameCallback { frame in
 for message in frame.embeddedMessages {
      if let seiMessage = message as? IVSUserDataUnregisteredSEIMessage {
          let seiMessageData = seiMessage.data
          let seiMessageUUID = seiMessage.UUID

          // interpret the message's data based on the UUID
      }
 }
}
```

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、配信を停止し、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

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、パブリッシャーが複数の異なるビデオの品質レイヤーを送信し、サブスクライバーがそれらのレイヤーを動的または手動で設定できるようにす

る IVS リアルタイムのストリーミング機能です。この機能は、「ストリーミング最適化」ドキュメントで詳しく説明されています。

レイヤードエンコーディングの設定 (パブリッシャー)

パブリッシャーとして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有効にするには、インスタンス化時に IVSLocalStageStream に次の設定を追加します。

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

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

// Other Stage implementation code
```

ビデオ設定で設定した解像度に応じて、「ストリーミングの最適化」の「デフォルトレイヤー、品質、フレームレート」セクションで定義されているように、設定された数のレイヤーがエンコードされて送信されます。

また、必要に応じて、サイマルキャスト設定内から個々のレイヤーを設定できます。

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

let layers = [
    IVSStagePresets.simulcastLocalLayer().default720(),
    IVSStagePresets.simulcastLocalLayer().default180()
]

try config.simulcast.setLayers(layers)

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

// Other Stage implementation code
```

または、最大で 3 つのレイヤー用に独自のカスタムレイヤー設定を作成することもできます。空のアレイを指定するか、値を指定しない場合、上記のデフォルトが使用されます。レイヤーは、次の必須プロパティセッターで説明されています。

- `setSize: CGSize;`

- setMaxBitrate: integer;
- setMinBitrate: integer;
- setTargetFramerate: float;

プリセットから、個々のプロパティを上書きするか、まったく新しい設定を作成できます。

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

let customHiLayer = IVSStagePresets.simulcastLocalLayer().default720()
try customHiLayer.setTargetFramerate(15)

let layers = [
    customHiLayer,
    IVSStagePresets.simulcastLocalLayer().default180()
]

try config.simulcast.setLayers(layers)

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

// Other Stage implementation code
```

個々のレイヤーを設定するときにトリガーできる最大値、制限、エラーについては、SDK リファレンスドキュメントを参照してください。

レイヤードエンコーディングの設定 (サブスクライバー)

サブスクライバーとして、レイヤードエンコーディングを有効にするために必要なものはありません。パブリッシャーがサイマルキャストレイヤーを送信している場合、デフォルトでサーバーによってレイヤー間で動的に適応され、サブスクライバーのデバイスおよびネットワークの状態に基づいて最適な品質が選択されます。

あるいは、パブリッシャーが送信している明示的なレイヤーを選択するには、以下に説明するいくつかのオプションがあります。

オプション 1: 初期レイヤー品質の選択

subscribeConfigurationForParticipant 戦略を使用すると、サブスクライバーとして受信する初期レイヤーを選択できます。

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration {
    let config = IVSSubscribeConfiguration()

    config.simulcast.initialLayerPreference = .lowestQuality

    return config
}
```

デフォルトでは、サブスクライバーは常に最初に最低品質のレイヤーが送信されます。これにより、徐々に最高品質のレイヤーにまで拡大します。エンドユーザーの帯域幅の消費量が最適化され、ビデオ再生に最適な時間が実現されるため、より貧弱なネットワーク上のユーザーに対して初期ビデオフリーズが軽減されます。

これらのオプションは InitialLayerPreference で利用できます。

- lowestQuality — サーバーは、最初に最低品質のビデオレイヤーを配信します。帯域幅の消費とメディアの時間が最適化されます。品質はビデオのサイズ、ビットレート、フレームレートの組み合わせとして定義されます。例えば、720p ビデオは 1080p ビデオよりも品質が低くなります。
- highestQuality — サーバーは、最初に最高品質のビデオレイヤーを配信します。品質が最適化されますが、メディアの時間が長くなる場合があります。品質はビデオのサイズ、ビットレート、フレームレートの組み合わせとして定義されます。例えば、1080p ビデオは 720p ビデオよりも高品質です。

注: 初期レイヤー設定 (initialLayerPreference の呼び出し) を反映させるには、再サブスクライブが必要です。これらの更新はアクティブなサブスクリプションに適用されないためです。

オプション 2: ストリームに優先されるレイヤー

preferredLayerForStream 戦略メソッドを使用すると、ストリームの開始後にレイヤーを選択できます。この戦略メソッドは参加者とストリーム情報を受け取るため、参加者ごとにレイヤーを選択できます。このメソッドは、ストリームのレイヤーが変化したとき、参加者の状態が変化したとき、またはホストアプリケーションが戦略を更新したときなど、特定のイベントに応じて SDK が呼び出します。

この戦略メソッドは IVSRemoteStageStreamLayer オブジェクトを返します。これは次のいずれかになります。

- IVSRemoteStageStream.layers によって返されるレイヤーオブジェクトなど。

- レイヤーを選択せず、動的適応が優先されることを示す null。

例えば、次の戦略ではユーザーが常に最低品質のビデオレイヤーを選択するようにします。

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, preferredLayerFor
 stream: IVSRemoteStageStream) -> IVSRemoteStageStreamLayer? {
    return stream.lowestQualityLayer
}
```

レイヤーの選択をリセットして動的適応に戻るには、戦略で null または未定義を返します。この例では、appState はホストアプリケーションの状態を表すプレースホルダー変数です。

```
func stage(_ stage: IVSStage, participant: IVSParticipantInfo, preferredLayerFor
 stream: IVSRemoteStageStream) -> IVSRemoteStageStreamLayer? {
    If appState.isAutoMode {
        return nil
    } else {
        return appState.layerChoice
    }
}
```

オプション 3: RemoteStageStream レイヤーヘルパー

IVSRemoteStageStream には、レイヤーの選択について決定し、対応する選択をエンドユーザーに表示するために使用できるいくつかのヘルパーがあります。

- レイヤーイベント — IVSStageRenderer に加え、IVSRemoteStageStreamDelegate にはレイヤーおよびサイマルキャストの適応変更を伝えるイベントがあります。
  - func stream(_ stream: IVSRemoteStageStream, didChangeAdaption adaption: Bool)
  - func stream(_ stream: IVSRemoteStageStream, didChange layers: [IVSRemoteStageStreamLayer])
  - func stream(_ stream: IVSRemoteStageStream, didSelect layer: IVSRemoteStageStreamLayer?, reason: IVSRemoteStageStream.LayerSelectedReason)
- レイヤーメソッド — IVSRemoteStageStream には、ストリームおよび提示されるレイヤーに関する情報を取得するために使用できるいくつかのヘルパーメソッドがあります。これらのメソッドは、preferredLayerForStream 戦略で提供されるリモートストリームに加え、func

`stage(_ stage: IVSStage, participant: IVSParticipantInfo, didAdd streams: [IVSStageStream])` を介して配信されるリモートストリームで利用できます。

- `stream.layers`
- `stream.selectedLayer`
- `stream.lowestQualityLayer`
- `stream.highestQualityLayer`
- `stream.layers(with: IVSRemoteStageStreamLayerConstraints)`

詳細については、「SDK リファレンスドキュメント」の「IVSRemoteStageStream」クラスを参照してください。LayerSelected の理由として UNAVAILABLE が返された場合、これはリクエストされたレイヤーが選択できなかったことを示します。代わりにベストエフォートの選択が行われ、ストリームの安定性を維持するために、通常は低品質のレイヤーが選択されます。

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、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 が管理するカメラは、熱の発生とエネルギー消費量を最小限に抑えるために、その解像度とフレームレート (1 秒あたりフレーム数、または FPS) を最適化します。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。

IVS iOS Broadcast SDK の既知の問題と回避策 | Real-Time Streaming

このドキュメントでは、Amazon IVS Real-Time Streaming iOS Broadcast SDK を使用する際に発生する可能性のある既知の問題の一覧を示し、可能な回避策を提案します。

- 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配信中でも、サブスクライバーが受け取る配信状態が `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。

- 参加者が配信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: 再配信/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、音声を配信停止にしてから再配信して問題を解決できます。

IVS iOS Broadcast SDK でのエラー処理 | Real-Time Streaming

このセクションでは、エラー状態の概要、IVS Real-Time Streaming iOS 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る処理について説明します。

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、配信/サブスクライブ状態に関連しており、配信/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、Swift 例外 (error code = 1000 および IVSBroadcastErrorIsFatalKey = YES) 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、Swift 例外 (error code = 1001 および IVSBroadcastErrorIsFatalKey = YES) 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 `stage(didChange connectionState, withError error)` を呼び出すと、error code = 1026 および IVSBroadcastErrorIsFatalKey = YES 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 `stage(didChange connectionState, withError error)` を呼び出すと、error code = 1300 および IVSBroadcastErrorIsFatalKey = YES 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 `stage(didChange connectionState, withError error)` を呼び出すと、error code = 1300 および IVSBroadcastErrorIsFatalKey 値= YES 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。

配信/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。再試行中の配信/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 `IVSErrorDelegate:didEmitError` を呼び出し、関連するエラーコードおよび `IVSBroadcastErrorIsFatalKey == NO` を出力します。

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

配信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、配信/サブスクライブ操作を再試行します。再試行中の配信/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 `didEmitError` を呼び出すと、error code = 1400 および IVSBroadcastErrorIsFatalKey = NO 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: 混合デバイス

混合デバイスは、複数の入力ソースを取得して単一の出力を生成するオーディオおよびビデオデバイスです。混合デバイスは、複数の画面上 (ビデオ) の要素およびオーディオトラックを定義および管理できる強力な機能です。カメラ、マイク、スクリーンキャプチャ、アプリで生成されたオーディオとビデオなど、複数のソースからのビデオとオーディオを組み合わせることができます。移行機能を使用して IVS にストリーミングするビデオ周辺にこれらのソースを移動させて、ストリーミングの途中でソースに追加およびソースを削除することができます。

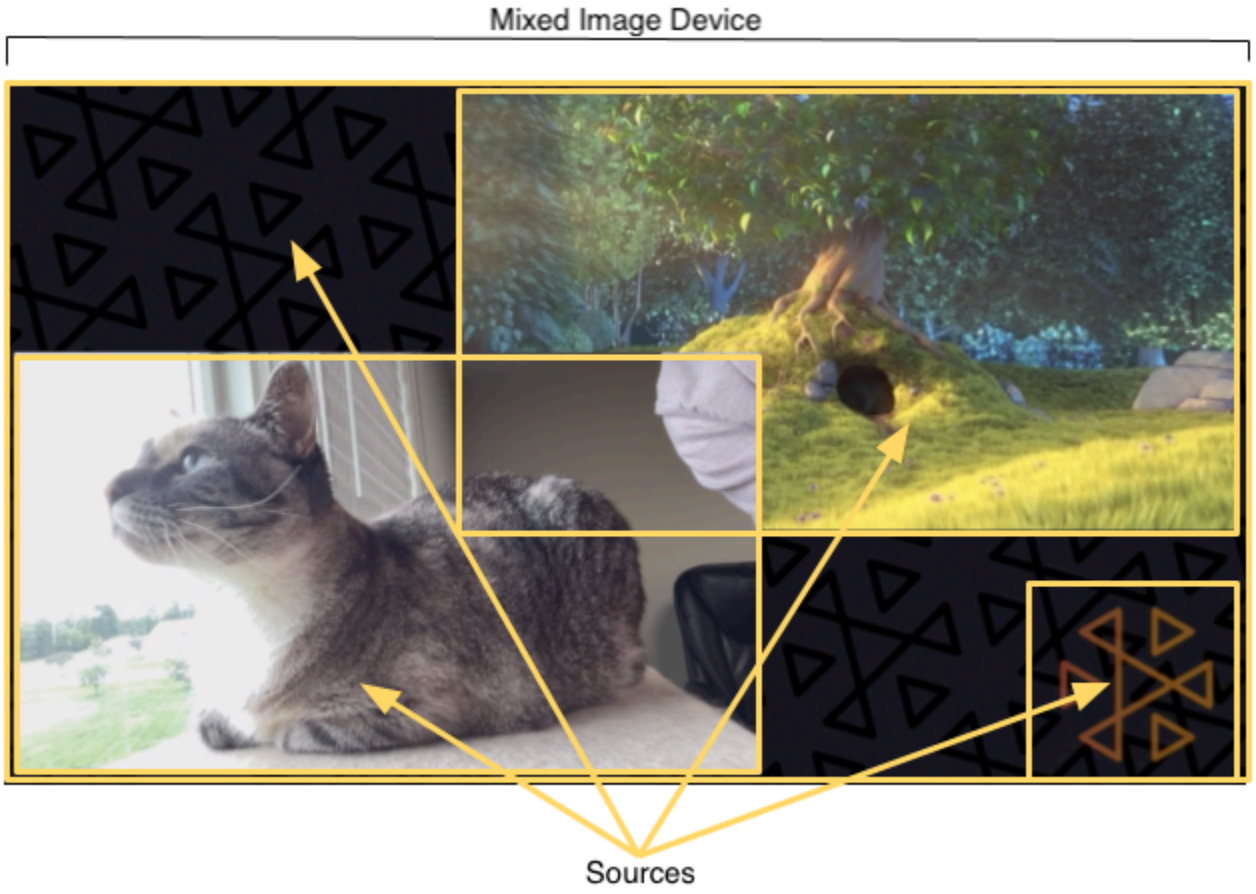
混合デバイスは画像およびオーディオの形式で存在します。混合イメージデバイスを作成するには、次のものを呼び出します。

Android の`DeviceDiscovery.createMixedImageDevice()`

iOS の `IVSDeviceDiscovery.createMixedImageDevice()`

返されたデバイスは、他のデバイスと同様に `BroadcastSession` (低レイテンシーストリーミング) または `Stage` (リアルタイムストリーミング) にアタッチできます。

用語



言葉	説明
デバイス	オーディオまたは画像入力を生成するハードウェアまたはソフトウェアコンポーネント。デバイスの例としては、マイク、カメラ、Bluetooth ヘッドセット、画面キャプチャやカスタム画像入力などの仮想デバイスがあります。
混合デバイス	他の Device と同様に、Device は BroadcastSession にアタッチできるますが、Source オブジェクトの追加を可能にする追加の API を使用します。混合デバイスにはオーディオまたは画像を複合する内部ミキサーがあり、単一の出力オーディオおよびイメージストリームを生成します。 混合デバイスはオーディオまたは画像のいずれかの形式で存在します。
混合デバイスの設定	混合デバイスの設定オブジェクト。混合イメージデバイスの場合、寸法やフレームレートなどのプロパティが設定されます。混合オーディオデバイスの場合、チャンネル数が設定されます。

言葉	説明
ソース	画面上のビジュアルエレメントの位置と、オーディオミックス内のオーディオトラックのプロパティを定義するコンテナです。混合デバイスは、ゼロ以上のソースで設定できます。ソースには、ソースのメディアの使用方法に影響する設定が与えられます。上のイメージには、4 つのイメージソースが示されています。 • 左下、カメラ入力あり • 右上にムービー入力あり • 右下に Amazon IVS の口ゴが入っています • フルスクリーンのバックグラウンドイメージ
ソース設定	混合デバイスに入るソースの設定オブジェクト。完全な設定オブジェクトは以下のように表記されています。
トランジション	スロットを新しい位置に移動したり、そのプロパティの一部を変更するには、`MixedDevice.transitionToConfiguration()` を使用します。このメソッドには次のものがあります。 • ソースの次の状態を表す新規のソース設定。 • ビデオのタイムラインを基準にして、アニメーションにかかる時間を指定するデュレーション。デュレーションを 0 に設定すると、ミックスされる次のフレームでトランジションが行われます。 • アニメーションが完了したことを通知するオプションのコールバック。コールバックは、アニメーションを連鎖させるのに便利です。

混合オーディオデバイス

設定

Android の`MixedAudioDeviceConfiguration`

iOS の `IVSMixedAudioDeviceConfiguration`

名前	型	説明
channels	整数	オーディオミキサーからの出力チャネルの数。有効な値:1、2。1 はモノラルオーディオで、2 はステレオオーディオです。デフォルト: 2。

ソース設定

Android のMixedAudioDeviceSourceConfiguration

iOS の IVSMixedAudioDeviceSourceConfiguration

名前	型	説明
gain	浮動小数点数	オーディオゲイン。これは乗数であるため、1 を超える値であればゲインが増加し、1 より小さい値であれば減少します。有効な値: 0 ~ 2。デフォルト: 1。

混合イメージデバイス

設定

Android のMixedImageDeviceConfiguration

iOS の IVSMixedImageDeviceConfiguration

名前	型	説明
size	Vec2	ビデオキャンバスのサイズ。
targetFramerate	整数	混合デバイスの 1 秒あたりのターゲットフレーム数。平均として、この値を満たす必要がありますが、特定の状況 (CPU や GPU の負荷の高さなど) でシステムのフレーム数が低下する場合があります。
transparencyEnabled	ブール値	イメージソース設定で alpha プロパティを使用してブレンドできます。これを true に設定すると、メ

名前	型	説明
		モリおよび CPU の消費量が増加します。デフォルト: `false`。

ソース設定

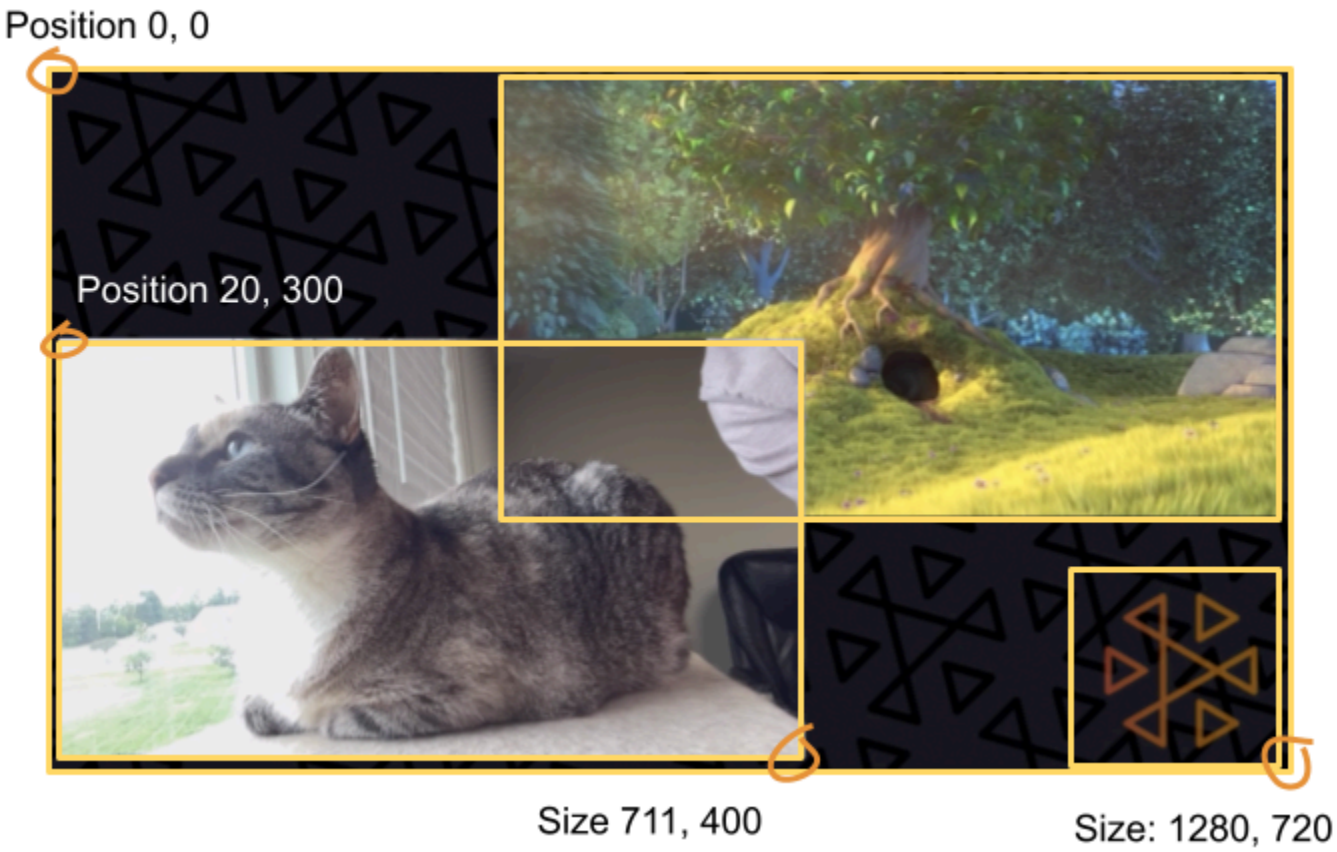
Android の`MixedImageDeviceSourceConfiguration`

iOS の `IVSMixedImageDeviceSourceConfiguration`

名前	型	説明
`alpha`	浮動小数点数	スロットのアルファ。これは、画像内のアルファ値との乗法です。有効な値: 0 ～ 1。0 は完全に透明で、1 は完全に不透明です。デフォルト: 1。
`aspect`	AspectMode	スロットでレンダリングされたイメージのアスペクト比モード。有効な値: • `Fill` — 画像のアスペクト比を維持しますが、スロットを埋めます。必要に応じて画像がトリミングされます。 • `Fit` — 画像のアスペクト比を維持しますが、画像全体をスロットに収めます。必要に応じて、スロットにレターボックスまたはピラーボックスが付いている場合があります。値が設定されている場合、レター/ピラーボックスは `fillColor` の中に位置されます。それ以外の場合は透明になります (画像の背後にあるキャンバスの色が黒の場合、黒で表示される場合があります)。 • None — イメージのアスペクト比を維持しないでください。イメージは、スロットの寸法に合わせてスケールされます。 デフォルト: `Fit`

名前	型	説明
fillColor	Vec4	スロットおよび画像のアスペクト比が一致しない場合に aspect Fit と使用する塗りつぶしカラー。フォーマットは (赤、緑、青、アルファ) です。有効な値 (チャネルごとに): 0 ～ 1。デフォルト: (0、0、0、0)。
position	Vec2	キャンバスの左上隅からの相対で、スロット位置 (ピクセル) です。スロットの原点も左上です。
size	Vec2	スロットのサイズ (ピクセル単位)。この値を設定すると、matchCanvasSize も false に設定されます。デフォルト: (0、0)。ただし、matchCanvasSize のデフォルトは true であるため、スロットのレンダリングサイズはキャンバスサイズであり、(0、0) ではありません。
zIndex	浮動小数点数	スロットの相対的な順序。zIndex 値が高いスロットは、zIndex 値が低いスロットの上に描画されます。

混合イメージデバイスの作成と設定



ここでは、このガイドの冒頭にあるシーンに似たシーンを作成し、次の 3 つの画面上の要素を使用します。

- カメラの左下のスロット。
- ロゴオーバーレイ用の右下のスロット。
- 映画の右上のスロット。

キャンバスの原点は左上隅で、これはスロットでも同じであることに注意してください。したがって、スロットを (0、0) に配置すると、スロット全体が見えるように左上隅に配置されます。

iOS

```
let deviceDiscovery = IVSDeviceDiscovery()
let mixedImageConfig = IVSMixedImageDeviceConfiguration()
mixedImageConfig.size = CGSize(width: 1280, height: 720)
try mixedImageConfig.setTargetFramerate(60)
mixedImageConfig.isTransparencyEnabled = true
```

```
let mixedImageDevice = deviceDiscovery.createMixedImageDevice(with: mixedImageConfig)

// Bottom Left
let cameraConfig = IVSMixedImageDeviceSourceConfiguration()
cameraConfig.size = CGSize(width: 320, height: 180)
cameraConfig.position = CGPoint(x: 20, y: mixedImageConfig.size.height -
 cameraConfig.size.height - 20)
cameraConfig.zIndex = 2
let camera = deviceDiscovery.listLocalDevices().first(where: { $0 is IVSCamera }) as?
 IVSCamera
let cameraSource = IVSMixedImageDeviceSource(configuration: cameraConfig, device:
 camera)
mixedImageDevice.add(cameraSource)

// Top Right
let streamConfig = IVSMixedImageDeviceSourceConfiguration()
streamConfig.size = CGSize(width: 640, height: 320)
streamConfig.position = CGPoint(x: mixedImageConfig.size.width -
 streamConfig.size.width - 20, y: 20)
streamConfig.zIndex = 1
let streamDevice = deviceDiscovery.createImageSource(withName: "stream")
let streamSource = IVSMixedImageDeviceSource(configuration: streamConfig, device:
 streamDevice)
mixedImageDevice.add(streamSource)

// Bottom Right
let logoConfig = IVSMixedImageDeviceSourceConfiguration()
logoConfig.size = CGSize(width: 320, height: 180)
logoConfig.position = CGPoint(x: mixedImageConfig.size.width - logoConfig.size.width -
 20,
                              y: mixedImageConfig.size.height - logoConfig.size.height
 - 20)
logoConfig.zIndex = 3
let logoDevice = deviceDiscovery.createImageSource(withName: "logo")
let logoSource = IVSMixedImageDeviceSource(configuration: logoConfig, device:
 logoDevice)
mixedImageDevice.add(logoSource)
```

Android

```
val deviceDiscovery = DeviceDiscovery(this /* context */)
val mixedImageConfig = MixedImageDeviceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(1280f, 720f))
```

```
    setTargetFramerate(60)
    setEnableTransparency(true)
}
val mixedImageDevice = deviceDiscovery.createMixedImageDevice(mixedImageConfig)

// Bottom Left
val cameraConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(320f, 180f))
    setPosition(BroadcastConfiguration.Vec2(20f, mixedImageConfig.size.y - size.y -
 20))
    setZIndex(2)
}
val camera = deviceDiscovery.listLocalDevices().firstNotNullOf { it as? CameraSource }
val cameraSource = MixedImageDeviceSource(cameraConfig, camera)
mixedImageDevice.addSource(cameraSource)

// Top Right
val streamConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(640f, 320f))
    setPosition(BroadcastConfiguration.Vec2(mixedImageConfig.size.x - size.x - 20,
 20f))
    setZIndex(1)
}
val streamDevice = deviceDiscovery.createImageInputSource(streamConfig.size)
val streamSource = MixedImageDeviceSource(streamConfig, streamDevice)
mixedImageDevice.addSource(streamSource)

// Bottom Right
val logoConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(320f, 180f))
    setPosition(BroadcastConfiguration.Vec2(mixedImageConfig.size.x - size.x - 20,
 mixedImageConfig.size.y - size.y - 20))
    setZIndex(1)
}
val logoDevice = deviceDiscovery.createImageInputSource(logoConfig.size)
val logoSource = MixedImageDeviceSource(logoConfig, logoDevice)
mixedImageDevice.addSource(logoSource)
```

ソースの削除

ソースを削除するには、削除する Source オブジェクトで MixedDevice.remove を呼び出します。

トランジションのあるアニメーション

トランジション方法では、ソースの構成が新しい構成に置き換えられます。この置換は、デュレーションを 0 より高く秒単位で設定することで、時間の経過とともにアニメートできます。

アニメーション化できるプロパティ

スロット構造のすべてのプロパティをアニメートできるわけではありません。Float タイプに基づくすべてのプロパティはアニメーション化できます。その他のプロパティは、アニメーションの開始時または終了時に有効になります。

名前	アニメーション化の可否	インパクトポイント
`Audio.gain`	はい	Interpolated
`Image.alpha`	はい	Interpolated
`Image.aspect`	いいえ	修了
`Image.fillColor`	はい	Interpolated
`Image.position`	はい	Interpolated
`Image.size`	はい	Interpolated
`Image.zIndex` 注: `zIndex` は 2D 平面を 3D 空間内で移動するため、アニメーションの途中で 2 つの平面が交差したときにトランジションが発生します。これは計算できますが、開始A値と終了 `zIndex` 値によって異なります。トランジションをよりスムーズにするために、これを `alpha` と組み合わせます。	はい	不明

シンプルな例

以下では、上記の混合イメージデバイスの作成と設定で定義された設定を使用したフルスクリーンのカメラテイクオーバーの例が示されます。これは 0.5 秒でアニメーション化されます。

iOS

```
// Continuing the example from above, modifying the existing cameraConfig object.
cameraConfig.size = CGSize(width: 1280, height: 720)
cameraConfig.position = CGPoint.zero
cameraSource.transition(to: cameraConfig, duration: 0.5) { completed in
    if completed {
        print("Animation completed")
    } else {
        print("Animation interrupted")
    }
}
```

Android

```
// Continuing the example from above, modifying the existing cameraConfig object.
cameraConfig.setSize(BroadcastConfiguration.Vec2(1280f, 720f))
cameraConfig.setPosition(BroadcastConfiguration.Vec2(0f, 0f))
cameraSource.transitionToConfiguration(cameraConfig, 500) { completed ->
    if (completed) {
        print("Animation completed")
    } else {
        print("Animation interrupted")
    }
}
```

ブロードキャストのミラーリング

ブロードキャストでアタッチされた画像デバイスを以下の方向にミラーリングする場合	以下の条件には負の値を使用します
水平方向	スロットの幅
垂直方向	スロットの高さ
水平方向と垂直方向の両方	スロットの幅と高さの両方

ミラーリング時にスロットを正しい位置に配置するには、同じ値で位置を調整する必要があります。

以下は、ブロードキャストを水平方向と垂直方向にミラーリングする例です。

iOS

水平ミラーリング:

```
let cameraSource = IVSMixedImageDeviceSourceConfiguration()
cameraSource.size = CGSize(width: -320, height: 720)
// Add 320 to position x since our width is -320
cameraSource.position = CGPoint(x: 320, y: 0)
```

垂直ミラーリング:

```
let cameraSource = IVSMixedImageDeviceSourceConfiguration()
cameraSource.size = CGSize(width: 320, height: -720)
// Add 720 to position y since our height is -720
cameraSource.position = CGPoint(x: 0, y: 720)
```

Android

水平ミラーリング:

```
val cameraConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(-320f, 180f))
   // Add 320f to position x since our width is -320f
    setPosition(BroadcastConfiguration.Vec2(320f, 0f))
}
```

垂直ミラーリング:

```
val cameraConfig = MixedImageDeviceSourceConfiguration().apply {
    setSize(BroadcastConfiguration.Vec2(320f, -180f))
    // Add 180f to position y since our height is -180f
    setPosition(BroadcastConfiguration.Vec2(0f, 180f))
}
```

注: このミラーリングは `ImagePreviewView` (Android) や `IVSImagePreviewView` (iOS) の `setMirrored` 方法とは異なります。このメソッドはデバイス上のローカルプレビューにのみ影響し、ブロードキャストには影響しません。

IVS Broadcast SDK: トークン交換 | リアルタイムストリーミング

トークン交換を使用すると、参加者が再接続することなく、参加者トークン機能をアップグレードまたはダウングレードし、モバイルブロードキャスト SDK 内のトークン属性を更新できます。これは、参加者がサブスクライブ専用機能から開始し、後でパブリッシュ機能が必要になる共同ホスティングなどのシナリオで便利です。

制限:

- トークン交換は、キーペアを使用してサーバーで作成されたトークンでのみ機能します。CreateParticipantToken API を介して作成されたトークンでは機能しません。
- トークン交換を使用して、サーバーサイドコンポジションレイアウト (featuredParticipantAttribute や participantOrderAttribute など) を駆動する属性を変更した場合、アクティブなコンポジションのレイアウトは、参加者が再接続するまで更新されません。

トークンの交換

トークンの交換は簡単です。`Stage` / `IVSStage` オブジェクト上の `exchangeToken` API を呼び出して新しいトークンを提供します。新しいトークンの `capabilities` が以前のトークンのものと異なる場合、新しいトークンの機能は直ちに評価されます。例えば、前のトークンに `publish` 機能がなく、新しいトークンにその機能がある場合、公開用のステージ戦略関数が呼び出され、ホストアプリケーションが新しい機能を使用してすぐに公開するか、待機するかを決定できるようになります。削除された機能も同様です。前のトークンに `publish` 機能があり、新しいトークンにその機能がない場合、参加者は公開用のステージ戦略関数を呼び出さずにすぐに公開を解除します。

トークンを交換する場合、以前のトークンと新しいトークンのペイロードフィールドの値が同じである必要があります。

- `topic`
- `resource`
- `jti`
- `whip_url`
- `events_url`

これらのフィールドはイミュータブルです。イミュータブルなフィールドを変更するトークンを交換すると、SDK は直ちに交換を拒否します。

残りのフィールドは、次のように変更できます。

- attributes
- capabilities
- user
- _id
- iat
- exp

iOS

```
let stage = try IVSStage(token: originalToken, strategy: self)
stage.join()
stage.exchangeToken(newToken)
```

Android

```
val stage = Stage(context, originalToken, strategy)
stage.join()
stage.exchangeToken(newToken)
```

更新の受信

StageRenderer / IVSStageRenderer の関数は、userId または attributes を更新するトークンを交換する既に公開済みのリモート参加者に関する更新を受信します。まだ公開していないリモート参加者は、最終的に公開された場合、既存の onParticipantJoined / participantDidJoin レンダラー関数を介して更新済みの userId と attributes を持ちます

iOS

```
class MyStageRenderer: NSObject, IVSStageRenderer {
    func stage(_ stage: IVSStage, participantMetadataDidUpdate participant:
 IVSParticipantInfo) {
        // participant will be a new IVSParticipantInfo instance with updated
 properties.
```

```
    }
}
```

Android

```
private val stageRenderer = object : StageRenderer {
    override fun onParticipantMetadataUpdated(stage: Stage, participantInfo:
 ParticipantInfo) {
        // participantInfo will be a new ParticipantInfo instance with updated
 properties.
    }
}
```

更新の可視性

参加者がトークンを交換して userId または attributes を更新する場合、これらの変更の可視性は現在の公開状態によって異なります。

- 参加者が公開していない場合: 更新はサイレントに処理されます。最終的に公開されると、すべての SDK は最初の公開イベントの一部として更新済みの userId と attributes を受け取ります。
- 参加者が既に公開している場合: 更新は直ちにブロードキャストされます。ただし、通知を受け取るのはモバイル SDK v1.37.0 以降のみです。ウェブ SDK、古いモバイル SDK、サーバーサイドコンポジションの参加者は、参加者が公開を解除して再公開するまで変更を表示されません。

次の表はサポートのマトリックスを示します。

参加者の状態	オブザーバー: モバイル SDK 1.37.0 以降	オブザーバー: 古いモバイル SDK、ウェブ SDK、サーバーサイドコンポジション
未発行 (その後開始)	# 表示可能 (参加者が参加したイベントを通じて公開)	# 表示可能 (参加者が参加したイベントを通じて公開)
発行済み (再発行なし)	# 表示可能 (参加者メタデータ更新済みイベントを介して即時)	# 表示不可

参加者の状態	オブザーバー: モバイル SDK 1.37.0 以降	オブザーバー: 古いモバイル SDK、ウェブ SDK、サーバーサイドコンポジション
発行済み (発行解除および再発行)	# 表示可能 (参加者メタデータ更新済みイベントを介して即時)	## 最終的に表示可能 (参加者が参加したイベントを介して再公開)

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) シーンのようにシンプルにすることや、カメラに加工フィルターを追加するなど、アプリケーションが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にカスタム画像入力ソースを使用する場合 (カメラアクセスを必要とするビューティーフィルターライブラリを使用するなど)、Broadcast SDK はカメラの管理をしなくなります。代わりに、アプリケーションはカメラのライフサイクルを正しく処理する責任があります。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` を取得します。

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

この Surface は、Camera2、OpenGL ES、その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 (加工フィルターライブラリなど) には、レンダリングする外部 Surface をアプリケーションで指定できるメソッドが用意されています。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、ストリーミングするブロードキャストセッションに、処理されたフレームを出力できます。

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 (加工フィルターなど) 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: カスタムオーディオソース | リアルタイムストリーミング

注: このガイドは、IVS リアルタイムストリーミング Android Broadcast SDK にのみ適用されます。iOS とウェブの SDK に関する情報は、今後公開される予定です。

カスタム音声入力ソースを使用すると、アプリケーションはデバイスの内蔵マイクに制限されるのではなく、ブロードキャスト SDK に独自の音声入力を提供できます。カスタムオーディオソースを使用すると、アプリケーションは処理された音声への工フェクトの適用とストリーミング、複数のオーディオストリームのミックス、サードパーティーのオーディオ処理ライブラリとの統合を行うことができます。

カスタムオーディオ入力ソースを使用する場合、ブロードキャスト SDK ではマイクが直接管理されなくなります。代わりに、アプリケーションがオーディオデータをキャプチャ、処理し、カスタムソースに送信します。

カスタムオーディオソースのワークフローは、以下のステップに従います。

1. オーディオ入力 — 指定されたオーディオ形式 (サンプルレート、チャネル、形式) でカスタムオーディオソースを作成します。
2. 処理 — オーディオ処理パイプラインからオーディオデータをキャプチャまたは生成します。
3. カスタムオーディオソース — appendBuffer( ) を使用してカスタムソースにオーディオバッファを送信します。
4. ステージ — LocalStageStream にラップして StageStrategy 経由でステージに公開します。
5. 参加者 — ステージ参加者は、処理された音声をリアルタイムで受け取ります。

Android

カスタムオーディオソースの作成

DeviceDiscovery セッションを作成した後、カスタム音声入力ソースを作成します。

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

// Create custom audio source with specific format
CustomAudioSource customAudioSource = deviceDiscovery.createAudioInputSource(
   2,  // Number of channels (1 = mono, 2 = stereo)
   BroadcastConfiguration.AudioSampleRate.RATE_48000,  // Sample rate
   AudioDevice.Format.INT16  // Audio format (16-bit PCM)
);
```

このメソッドは、未加工の PCM オーディオデータを受け入れる CustomAudioSource を返します。カスタムオーディオソースは、オーディオ処理パイプラインが生成するのと同じオーディオ形式で設定する必要があります。

サポートされるオーディオ形式

パラメータ	オプション	説明
チャンネル	1 (モノラル)、2 (ステレオ)	オーディオチャネルの数。
サンプルレート	RATE_16000、RATE_44100、RATE_48000	オーディオサンプルレート (Hz)。高品質には 48kHz が推奨されます。
形式	INT16、FLOAT32	オーディオサンプル形式。INT16 は 16 ビットの固定ポイント PCM です。FLOAT32 は 32 ビットの浮動小数点 PCM です。インターリーブ形式と Planar 平面形式の両方を使用できます。

オーディオデータの送信

オーディオデータをカスタムソースに送信するには、appendBuffer() メソッドを使用します。

```
// Prepare audio data in a ByteBuffer
ByteBuffer audioBuffer = ByteBuffer.allocateDirect(bufferSize);
audioBuffer.put(pcmAudioData);  // Your processed audio data

// Calculate the number of bytes
long byteCount = pcmAudioData.length;

// Submit audio to the custom source
// presentationTimeUs should be generated by and come from your audio source
int samplesProcessed = customAudioSource.appendBuffer(
   audioBuffer,
   byteCount,
   presentationTimeUs
);

if (samplesProcessed > 0) {
   Log.d(TAG, "Successfully submitted " + samplesProcessed + " samples");
} else {
   Log.w(TAG, "Failed to submit audio samples");
}

// Clear buffer for reuse
```

```
audioBuffer.clear();
```

重要な考慮事項:

- オーディオデータは、カスタムソースの作成時に指定された形式である必要があります。
- スムーズなオーディオ再生のために、タイムスタンプは一定間隔で増加し、オーディオソースによって提供される必要があります。
- ストリームのギャップを避けるため、オーディオを定期的に送信します。
- メソッドは、処理されたサンプルの数を返します (0 は失敗を示します) 。

ステージへの発行

CustomAudioSource を AudioLocalStageStream でラップし、StageStrategy から返します。

```
// Create the audio stream from custom source
AudioLocalStageStream audioStream = new AudioLocalStageStream(customAudioSource);

// Define your stage strategy
Strategy stageStrategy = new Strategy() {
   @NonNull
   @Override
   public List<LocalStageStream> stageStreamsToPublishForParticipant(
         @NonNull Stage stage,
         @NonNull ParticipantInfo participantInfo) {
      List<LocalStageStream> streams = new ArrayList<>();
      streams.add(audioStream);  // Publish custom audio
      return streams;
   }

   @Override
   public boolean shouldPublishFromParticipant(
         @NonNull Stage stage,
         @NonNull ParticipantInfo participantInfo) {
      return true;  // Control when to publish
   }

   @Override
   public Stage.SubscribeType shouldSubscribeToParticipant(
         @NonNull Stage stage,
         @NonNull ParticipantInfo participantInfo) {
```

```
        return Stage.SubscribeType.AUDIO_VIDEO;
    }
};

// Create and join the stage
Stage stage = new Stage(context, stageToken, stageStrategy);
```

完全な例: オーディオ処理の統合

オーディオ処理 SDK との統合を示す完全な例を次に示します。

```
public class AudioStreamingActivity extends AppCompatActivity {
    private DeviceDiscovery deviceDiscovery;
    private CustomAudioSource customAudioSource;
    private AudioLocalStageStream audioStream;
    private Stage stage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Configure audio manager
        StageAudioManager.getInstance(this)
            .setPreset(StageAudioManager.UseCasePreset.VIDEO_CHAT);

        // Initialize IVS components
        initializeIVSStage();

        // Initialize your audio processing SDK
        initializeAudioProcessing();
    }

    private void initializeIVSStage() {
        deviceDiscovery = new DeviceDiscovery(this);

        // Create custom audio source (48kHz stereo, 16-bit)
        customAudioSource = deviceDiscovery.createAudioInputSource(
            2,  // Stereo
            BroadcastConfiguration.AudioSampleRate.RATE_48000,
            AudioDevice.Format.INT16
        );

        // Create audio stream
```

```
        audioStream = new AudioLocalStageStream(customAudioSource);

        // Create stage with strategy
        Strategy strategy = new Strategy() {
            @NonNull
            @Override
            public List<LocalStageStream> stageStreamsToPublishForParticipant(
                    @NonNull Stage stage,
                    @NonNull ParticipantInfo participantInfo) {
                return Collections.singletonList(audioStream);
            }

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

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

        stage = new Stage(this, getStageToken(), strategy);
    }

    private void initializeAudioProcessing() {
        // Initialize your audio processing SDK
        // Set up callback to receive processed audio
        yourAudioSDK.setAudioCallback(new AudioCallback() {
            @Override
            public void onProcessedAudio(byte[] audioData, int sampleRate,
                                         int channels, long timestamp) {
                // Submit processed audio to IVS Stage
                submitAudioToStage(audioData, timestamp);
            }
        });
    }

    // The timestamp is required to come from your audio source and you
```

```
    // should not be generating one on your own, unless your audio source
    // does not provide one. If that is the case, create your own epoch
    // timestamp and manually calculate the duration between each sample
    // using the number of frames and frame size.

    private void submitAudioToStage(byte[] audioData, long timestamp) {
        try {
            // Allocate direct buffer
            ByteBuffer buffer = ByteBuffer.allocateDirect(audioData.length);
            buffer.put(audioData);

            // Submit to custom audio source
            int samplesProcessed = customAudioSource.appendBuffer(
                buffer,
                audioData.length,
                timestamp > 0 ? timestamp : System.nanoTime() / 1000
            );

            if (samplesProcessed <= 0) {
                Log.w(TAG, "Failed to submit audio samples");
            }

            buffer.clear();
        } catch (Exception e) {
            Log.e(TAG, "Error submitting audio: " + e.getMessage(), e);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (stage != null) {
            stage.release();
        }
    }
}
```

ベストプラクティス

オーディオ形式の整合性

送信するオーディオ形式が、カスタムソースの作成時に指定された形式と一致していることを確認します。

```
// If you create with 48kHz stereo INT16
customAudioSource = deviceDiscovery.createAudioInputSource(
   2, RATE_48000, INT16
);

// Your audio data must be:
// - 2 channels (stereo)
// - 48000 Hz sample rate
// - 16-bit interleaved PCM format
```

バッファ管理

ガベージコレクションを最小限に抑えるために ByteBuffers を直接使用して再利用します。

```
// Allocate once
private ByteBuffer audioBuffer = ByteBuffer.allocateDirect(BUFFER_SIZE);

// Reuse in callback
public void onAudioData(byte[] data) {
   audioBuffer.clear();
   audioBuffer.put(data);
   customAudioSource.appendBuffer(audioBuffer, data.length, getTimestamp());
   audioBuffer.clear();
}
```

タイミングと同期

オーディオをスムーズに再生するには、オーディオソースが提供するタイムスタンプを使用する必要があります。オーディオソースに独自のタイムスタンプがない場合は、独自の工ポックタイムスタンプを作成し、フレーム数とフレームサイズを使用して各サンプル間の期間を手動で計算します。

```
// "audioFrameTimestamp" should be generated by your audio source
// Consult your audio source’s documentation for information on how to get this
long timestamp = audioFrameTimestamp;
```

エラー処理

必ず appendBuffer( ) の戻り値を確認してください。

```
int samplesProcessed = customAudioSource.appendBuffer(buffer, count, timestamp);
```

```
if (samplesProcessed <= 0) {
   Log.w(TAG, "Audio submission failed - buffer may be full or format mismatch");
   // Handle error: check format, reduce submission rate, etc.
}
```

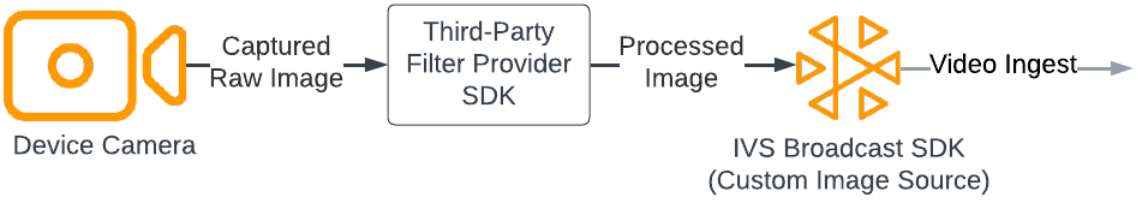
IVS Broadcast SDK: サードパーティーのカメラフィルター | リアルタイムストリーミング

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。

IVS Broadcast SDK で BytePlus を使用する

このドキュメントでは、IVS Broadcast SDK で BytePlus Effects SDK を使用する方法について説明します。

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);
```

IVS Broadcast SDK で DeepAR を使用する

このドキュメントでは、IVS Broadcast SDK で DeepAR SDK を使用する方法について説明します。

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。

VS Broadcast SDK で Snap を使用する

このドキュメントでは、IVS Broadcast SDK で Snap の Camera Kit SDK を使用する方法について説明します。

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。

Snap の Camera Kit SDK を IVS Real-Time Streaming 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. `index.css` を作成します。
5. 参加者を表示して設定します。
6. 接続されているカメラとマイクを表示します。
7. Camera Kit セッションを作成します。
8. レンズを取得し、レンズセレクターに入力します。
9. 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。
10. [レンズ] ドロップダウンに入力する関数を作成します。
11. 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。
12. `package.json` を作成します。
13. Webpack 設定ファイルを作成します。
14. HTTPS サーバーをセットアップしてテストします。

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

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。

```
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。

HTML

```
<!--
/*! 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/LowLatencyUserGuide/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 -->

  <!-- Display Local Participants -->

  <!-- Lens Selector -->

  <!-- Display Remote Participants -->

  <!-- Load All Desired Scripts -->
```

セットアップ要素を追加する

カメラ、マイク、およびレンズを選択し、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。

HTML

```
<!-- 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。

HTML

```
  <!-- 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) body および html 要素の終了タグを追加して、index.html の作成を完了します。

HTML

```
<!-- 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>
</body>
</html>
```

index.css を作成する

ページのスタイルを設定するための CSS ソースファイルを作成します。ステージを管理し、Snap の Camera Kit SDK と統合するためのロジックに焦点を当てるため、このコードについては説明しません。

CSS

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

html,
body {
  margin: 2rem;
  box-sizing: border-box;
  height: 100vh;
  max-height: 100vh;
  display: flex;
  flex-direction: column;
}

hr {
  margin: 1rem 0;
}

table {
  display: table;
}
```

```
canvas {
  margin-bottom: 1rem;
  background: green;
}

video {
  margin-bottom: 1rem;
  background: black;
  max-width: 100%;
  max-height: 150px;
}

.log {
  flex: none;
  height: 300px;
}

.content {
  flex: 1 0 auto;
}

.button {
  display: block;
  margin: 0 auto;
}

.local-container {
  position: relative;
}

.static-controls {
  position: absolute;
  margin-left: auto;
  margin-right: auto;
  left: 0;
  right: 0;
  bottom: -4rem;
  text-align: center;
}

.static-controls button {
  display: inline-block;
}
```

```
.hidden {
  display: none;
}

.participant-container {
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction: column;
  margin: 1rem;
}

video {
  border: 0.5rem solid #555;
  border-radius: 0.5rem;
}
.placeholder {
  background-color: #333333;
  display: flex;
  text-align: center;
  margin-bottom: 1rem;
}
.placeholder span {
  margin: auto;
  color: white;
}
#local-media {
  display: inline-block;
  width: 100vw;
}

#local-media video {
  max-height: 300px;
}

#remote-media {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: row;
  width: 100%;
}
```

```
#lens-selector {
  width: 100%;
  margin-bottom: 1rem;
}
```

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。次のコードブロックをコピーして stages.js に貼り付けることをお勧めします。その後、コードの内容を 1 つずつ確認すると、後続のセクションで何が起こっているか把握できます。

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

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');

let lensSelector = document.getElementById('lens-selector');
let session;
let availableLenses = [];

// 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_YOUR_API_TOKEN_HERE',
  });

  session = await cameraKit.createSession({ liveRenderTarget });
  const { lenses } = await cameraKit.lensRepository.loadLensGroups([
    'INSERT_YOUR_LENS_GROUP_ID_HERE',
  ]);

  availableLenses = lenses;
  populateLensSelector(lenses);

  const snapStream = liveRenderTarget.captureStream();

  lensSelector.addEventListener('change', handleLensChange);
  lensSelector.disabled = true;
  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();
  });
};

async function setCameraKitSource(session, mediaStream) {
  const source = createMediaStreamSource(mediaStream);
  await session.setSource(source);
  source.setTransform(Transform2D.MirrorX);
  session.play();
}

const populateLensSelector = (lenses) => {
  lensSelector.innerHTML = '<option selected disabled>Choose Lens</option>';

  lenses.forEach((lens, index) => {
    const option = document.createElement('option');
    option.value = index;
    option.text = lens.name || `Lens ${index + 1}`;
    lensSelector.appendChild(option);
  });
};

const handleLensChange = (event) => {
  const selectedIndex = parseInt(event.target.value);
  if (session && availableLenses[selectedIndex]) {
    session.applyLens(availableLenses[selectedIndex]);
  }
};

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(snapStream.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');
    lensSelector.disabled = false;
  } else {
    controls.classList.add('hidden');
    lensSelector.disabled = true;
  }
});

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();
```

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。

JavaScript

```
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */

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');

let lensSelector = document.getElementById('lens-selector');
let session;
let availableLenses = [];

// 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_YOUR_API_TOKEN_HERE',
  });

  session = await cameraKit.createSession({ liveRenderTarget });
```

レンズの取得とレンズセレクターへの入力

レンズを取得するには、Lens グループ ID のプレースホルダーを、Camera Kit デベロッパーポータルにある独自の ID に置き換えます。後で作成する `populateLensSelector()` 関数を使用して、レンズ選択ドロップダウンに入力します。

JavaScript

```
session = await cameraKit.createSession({ liveRenderTarget });
  const { lenses } = await cameraKit.lensRepository.loadLensGroups([
    'INSERT_YOUR_LENS_GROUP_ID_HERE',
  ]);

  availableLenses = lenses;
  populateLensSelector(lenses);
```

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();

  lensSelector.addEventListener('change', handleLensChange);
  lensSelector.disabled = true;
  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();
  });
};
```

[レンズ] ドロップダウンに入力する関数を作成する

次の関数を作成し、先ほど取得したレンズをレンズセレクターに入力します。レンズセレクターは、カメラフィードに適用するレンズをリストから選択できるページ上の UI 要素です。また、handleLensChange コールバック関数を作成して、[レンズ] ドロップダウンから選択したときに指定したレンズを適用することもできます。

JavaScript

```
const populateLensSelector = (lenses) => {
  lensSelector.innerHTML = '<option selected disabled>Choose Lens</option>';

  lenses.forEach((lens, index) => {
```

```
    const option = document.createElement('option');
    option.value = index;
    option.text = lens.name || `Lens ${index + 1}`;
    lensSelector.appendChild(option);
  });
};

const handleLensChange = (event) => {
  const selectedIndex = parseInt(event.target.value);
  if (session && availableLenses[selectedIndex]) {
    session.applyLens(availableLenses[selectedIndex]);
  }
};
```

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();
```

package.json を作成する

package.json を作成して、次の JSON 設定を追加します。このファイルは依存関係を定義するもので、コードをバンドルするためのスクリプトコマンドが含まれています。

JSON 設定

```
{
  "dependencies": {
    "@snap/camera-kit": "^0.10.0"
  },
  "name": "ivs-stages-with-snap-camerakit",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "build": "webpack"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "webpack": "^5.95.0",
    "webpack-cli": "^5.1.4"
```

```
  }
}
```

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、これまでに作成したコードがバンドルされ、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。テスト目的であれば、ローカルコンピュータから HTML と JavaScript を提供することができます。この例では、Python の http.server モジュールを使用します。

HTTPS サーバーのセットアップとテスト

コードをテストするには、HTTPS サーバーをセットアップする必要があります。ウェブアプリケーションの Snap Camera Kit SDK との統合をローカルで開発およびテストするために HTTPS サーバーを使用すると、オリジン間リソース共有 (CORS、Cross-Origin Resource Sharing) の問題を回避できます。

ターミナルを開き、この時点までのすべてのコードを作成したディレクトリに移動します。次のコマンドを実行して、自己署名 SSL/TLS 証明書とプライベートキーを生成します。

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes
```

これにより、key.pem (プライベートキー) と cert.pem (自己署名証明書) の 2 つのファイルが作成されます。https_server.py という名前の新しい Python ファイルを作成し、次のコードを追加します。

Python

```
import http.server
```

```
import ssl

# Set the directory to serve files from
DIRECTORY = '.'

# Create the HTTPS server
server_address = ('', 4443)
httpd = http.server.HTTPServer(
    server_address, http.server.SimpleHTTPRequestHandler)

# Wrap the socket with SSL/TLS
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain('cert.pem', 'key.pem')
httpd.socket = context.wrap_socket(httpd.socket, server_side=True)

print(f'Starting HTTPS server on https://localhost:4443, serving {DIRECTORY}')
httpd.serve_forever()
```

ターミナルを開き、https_server.py ファイルを作成したディレクトリに移動し、次のコマンドを実行します。

```
python3 https_server.py
```

これにより、https://localhost:4443 で HTTPS サーバーが起動し、現在のディレクトリからファイルが提供されます。cert.pem および key.pem ファイルが https_server.py ファイルと同じディレクトリにあることを確認します。

ブラウザを開き、https://localhost:4443 に移動します。これは自己署名 SSL/TLS 証明書であるため、お使いのウェブブラウザでは信頼されず、警告が表示されます。これはテスト目的のみであるため、警告をバイパスできます。次に、前に指定したスナップレンズの AR 効果が、カメラフィードに画面に表示されます。

Python の組み込み http.server モジュールと ssl モジュールを使用したこの設定は、ローカルでの開発およびテスト目的には適していますが、本番環境には推奨されません。この設定で使用される自己署名 SSL/TLS 証明書は、ウェブブラウザやその他のクライアントで信頼されていないため、ユーザーがサーバーにアクセスするとセキュリティ警告が表示されます。また、この例では Python の組み込み http.server モジュールと ssl モジュールを使用していますが、別の HTTPS サーバーソリューションを使用することもできます。

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。

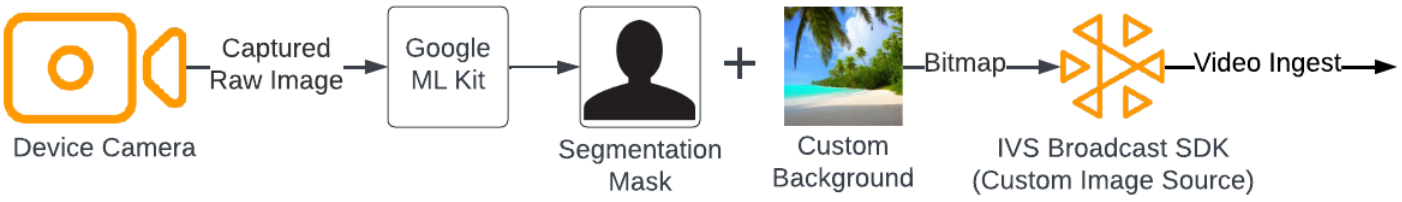
Java

```
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
```

IVS Broadcast SDK で背景の置換を使用する

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 Real-Time Streaming 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. `index.html` を作成します。
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. `app.js` を作成します。
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 Real-Time Streaming 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。

Volume Rocker カテゴリは、デバイスの物理ボリュームロッカーを介して使用または変更されたボリュームのタイプ (メディアボリュームまたはコールボリューム) を指します。これは、オーディオモードを切り替えるときにボリュームに影響することに注意してください。例えば、Video Chat プリセットの使用中にデバイスボリュームが最大値に設定されているとします。Subscribe Only プリセットに切り替えると、オペレーティングシステムとは異なるボリュームレベルになり、デバイスで大幅なボリューム変更が発生する可能性があります。

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。

iOS の既知の問題: このプリセットを使用してマイクをアタッチしないと、デバイススピーカーではなくイヤホンでオーディオが再生されます。このプリセットは、マイクと組み合わせてのみ使用してください。

Category	Android	iOS
エコーキャンセレーション	有効	有効

Category	Android	iOS
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。
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。

Category	Android	iOS
エコーキャンセレーション	Disabled	Disabled
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。

Category	Android	iOS
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。

Category	Android	iOS
エコーキャンセレーション	プラットフォームエコーキャンセレーションは無効になっていますが、 `StageAudioConfiguration.enableEchoCancellation` が true の場合、ソフトウェアエコーキャンセレーションが発生する可能性があります。	Disabled
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。	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。

Category	Android	iOS
	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。	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。

注: これらのオーディオ SDK メソッドを使用する場合、基盤となるオーディオセッションを誤って設定する可能性があります。例えば、iOS で `.allowBluetooth` オプションを `.playback` カテゴリと組み合わせて使用すると、無効なオーディオ設定が作成され、SDK はオーディオを録音または再生できません。これらのメソッドは、アプリケーションに特定のオーディオセッション要件が検証されている場合にのみ使用されるように設計されています。

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
```

iOS エコーキャンセレーション

iOS でのエコーキャンセレーションは、echoCancellationEnabled メソッドを使用して IVSStageAudioManager を介して個別に制御することもできます。このメソッドは、SDK が使用し、基盤となっている AVAudioEngine の入出力ノードで音声処理が有効になっているかどうかを制御します。このプロパティを手動で変更した場合の以下の影響を理解することは重要です。

- AVAudioEngine プロパティは、SDK のマイクがアクティブな場合にのみ使用されます。これは、入力ノードと出力ノードの両方で音声処理を同時に有効にするという iOS の要件のために必要です。通常、これは、IVSDeviceDiscovery から返されたマイクを使用して行われ、公開用に IVSLocalStageStream が作成されます。または、マイク自体に IVSAudioDeviceStatsCallback をアタッチすることで、公開に使用することなくマイクを有効にすることもできます。この代替アプローチは、IVS SDK のマイクの代わりにカスタムオーディオソースベースのマイクを使用しているときにエコーキャンセレーションが必要な場合に役立ちます。
- AVAudioEngine プロパティを有効にするには、.videoChat または .voiceChat のモードが必要です。別のモードをリクエストすると、iOS の基盤となるオーディオフレームワークが SDK と戦い、オーディオが失われます。
- AVAudioEngine を自動的に有効にすると、.allowBluetooth オプションが有効になります。

デバイスと iOS のバージョンによって動作が異なる場合があります。

iOS カスタムオーディオソース

カスタムオーディオソースは、`IVSDeviceDiscovery.createAudioSource` を使用して SDK で使用できます。ステージに接続する場合、IVS Real-Time Streaming Broadcast SDK は、SDK のマイクが使用されていなくても、音声再生用の内部 `AVAudioEngine` インスタンスを管理します。そのため、`IVSStageAudioManager` に提供される値は、カスタムオーディオソースによって提供されるオーディオと互換性がある必要があります。

公開に使用されるカスタムオーディオソースがマイクからの録音でホストアプリケーションで管理されている場合、上記のエコーキャンセレーション SDK は、SDK 管理のマイクがアクティブ化されない限り機能しません。この要件を回避する方法については、「iOS エコーキャンセレーション」を参照してください。

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 Real-Time Streaming 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 Real-Time Streaming 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 Real-Time Streaming 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	送信先への出力の試行が失敗しました (例: S3 バケットが見つからない、S3 バケットへのアクセスが拒否された、RTMP 送信先に既にストリームが存在する) 。
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	ステージの削除、1 つ以上の出力の失敗、またはその他の内部エラーにより、コンポジションパイプラインが失敗しました。
IVS 参加者の録画状態の変更	Recording Start	パブリッシャーがステージに接続し、S3 に記録されています。
IVS 参加者の録画状態の変更	Recording End	パブリッシャーがステージから切断され、残りのすべてのファイルが S3 に書き込まれました。
IVS 参加者の録画状態の変更	Recording Start Failure	パブリッシャーはステージに接続しますが、エラー (S3 バケットが見つからない場合やアクセスできない場合など) により記録が開始されません。このパブリッシャーのライブストリームは録画されません。

イベントタイプ	イベント	配信するタイミング
IVS 参加者の録画状態の変更	Recording End Failure	記録中にエラーが発生したため (S3 バケットが見つからない場合やアクセスできない場合など)、記録は失敗で終了します。一部のオブジェクトが、設定された保存場所に引き続き書き込まれることがあります。
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。
IVS Stage Update	参加者の公開エラー	参加者がステージに公開しようとしましたが失敗しました。
IVS Stage Update	参加者のレプリケーション開始	参加者のレプリケーションが開始されます。
IVS Stage Update	参加者のレプリケーション終了	参加者のレプリケーションが終了します。レプリケーションは、StopParticipantReplication API オペレーションによって終了することがあります。また、パブリッシャーが公開を停止した場合や、パブリッシャーが公開を停止し、再接続ウィンドウが期限切れになった場合にも終了することがあります。
IVS Stage Update	交換済みトークン	既存の参加者トークンは新しいトークンと交換されます。この交換により、トークン機能のアップグレードまたはダウングレード、およびトークン属性の更新が行われます。

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

送信先の失敗: このイベントは、送信先への出力の試行が失敗した場合 (S3 バケットが見つからなかった場合、S3 バケットへのアクセスが拒否された場合、または RTMP 送信先に対してストリームが既に存在する場合) 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 Failure",
     "stage_arn": "<stage-arn>",
     "id": "<Destination-id>",
     "error_code": "e.g., AccessDeniedException",
     "reason": "e.g., Access denied to S3 bucket. Please verify your bucket policy"
   }
}
```

次の表に、送信先障害イベントの `error_code` と `reason` 値、およびトラブルシューティングガイダンスを示します。

error_code	理由	トラブルシューティングのガイダンス
ResourceNotFoundException	S3 バケットが見つかりません。	S3 バケットが存在し、正しいリージョンにあることを確認します。

error_code	理由	トラブルシューティングのガイダンス
	バケットが存在することを確認してください。	
AccessDeniedException	S3 バケットへのアクセスが拒否されました。バケットポリシーを確認してください。	S3 バケットポリシーが IVS サービスに必要なアクセス許可を付与していることを確認します。
ConflictException	ストリームは既に存在します	同じ RTMP 送信先チャネルで他のブロードキャストがアクティブでないことを確認します。
InternalServerException	サービス内部エラー	オペレーションを再試行する。問題が解決しない場合は、AWS サポートまでお問い合わせください。

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>"
   }
}
```

セッションの失敗: このイベントは、ステージの削除、1 つ以上の出力の失敗、またはその他の内部エラーが原因でコンポジションパイプラインが失敗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": "Session Failure",
     "stage_arn": "<stage-arn>",
     "error_code": "e.g., DestinationFailure",
     "reason": "e.g. One or more outputs failed"
   }
}
```

次の表に、セッション障害イベントの error_code と reason 値、およびトラブルシューティングガイダンスを示します。

error_code	理由	トラブルシューティングのガイダンス
StageDeleted	ステージが削除されました	コンポジションを開始する前に、ステージが存在することを確認します。
DestinationFailure	1 つ以上の出力が失敗しました	個々の送信先エラーを確認します。
InternalServerException	サービス内部エラー	オペレーションを再試行する。問題が解決しない場合は、AWS サポートまでお問い合わせください。

例: 個々の参加者の録画状態の変更

Recording Start: このイベントは、パブリッシャーがステージに接続し、S3 に記録されているときに送信されます。

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:09:58Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording Start",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "bucket-name",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z"
   }
}
```

Recording End: このイベントは、パブリッシャーがステージから切断され、残りのすべてのファイルが S3 に書き込まれたときに送信されます。

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:19:04Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording End",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "bucket-name",
```

```
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z",
      "recording_duration_ms": 547327
   }
}
```

記録開始の失敗: このイベントは、パブリッシャーはステージに接続しましたが、エラー (S3 バケットが存在しない、または正しいリージョンにないなど) により、録画の開始に失敗しました。パブリッシャーのライブストリームは録画されません。

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:09:58Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording Start Failure",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "bucket-name",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z",
      "error_code": "e.g., AccessDeniedException",
      "reason": "e.g., Access denied to S3 bucket. Please verify your bucket policy"
   }
}
```

次の表に、記録開始の失敗イベントの error_code と reason 値、およびトラブルシューティングガイダンスを示します。

error_code	理由	トラブルシューティングのガイダンス
ResourceNotFoundException	S3 バケットが見つかりません。バケットが存在	S3 バケットが存在し、正しいリージョンにあることを確認します。

error_code	理由	トラブルシューティングのガイダンス
	することを確認してください。	
AccessDeniedException	S3 バケットへのアクセスが拒否されました。バケットポリシーを確認してください。	S3 バケットポリシーが IVS サービスに必要なアクセス許可を付与していることを確認します。
ValidationException	ビデオコーデックは録画ではサポートされていません	パブリッシャーがサポートされているビデオコーデックを使用していることを確認します。
InternalServerException	サービス内部エラー	オペレーションを再試行する。問題が解決しない場合は、AWS サポートまでお問い合わせください。

記録終了の失敗: このイベントは、録画中に発生したエラー (S3 バケットが見つからない場合やアクセスできない場合など) により、録画が失敗して終了したときに送信されます。一部のオブジェクトが、設定された保存場所に引き続き書き込まれることがあります。

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:19:04Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording End Failure",
      "participant_id": "xYz1c2d3e4f",
```

```
      "recording_s3_bucket_name": "bucket-name",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z",
      "recording_duration_ms": 547327,
      "error_code": "e.g., AccessDeniedException",
      "reason": "e.g., Access denied to S3 bucket. Please verify your bucket policy"
   }
}
```

次の表に、記録終了の失敗イベントの error_code と reason 値、およびトラブルシューティングガイダンスを示します。

error_code	理由	トラブルシューティングのガイダンス
ResourceNotFoundException	S3 バケットが見つかりません。バケットが存在することを確認してください。	S3 バケットが存在し、正しいリージョンにあることを確認します。
AccessDeniedException	S3 バケットへのアクセスが拒否されました。バケットポリシーを確認してください。	S3 バケットポリシーが IVS サービスに必要なアクセス許可を付与していることを確認します。
InternalServerException	サービス内部エラー	オペレーションを再試行する。問題が解決しない場合は、AWS サポートまでお問い合わせください。

個々の参加者の記録マージが有効で、ステージパブリッシャーがステージから切断してから再接続した場合、IVS は前のセッションと同じ S3 プレフィックスに記録しようとすることに注意してください。その結果、上記の例では recording_s3_key_prefix の session_id コンポーネントは detail で session_id フィールドとは異なる値を持つことができます。「フラグメント化された個々の参加者の記録をマージする」を参照してください。

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-ZyXwvu1T2s",
        "event_name": "Participant Published",
        "event_time": "2025-11-18T16:40:32Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "replica": true,
        "source_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij",
        "source_session_id": "st-sdfdfdfgdfgh"
    }
}
```

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-ZyXwvu1T2s",
        "event_name": "Participant Unpublished",
        "event_time": "2025-11-18T16:40:32Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "replica": true,
        "source_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij",
        "source_session_id": "st-sdfdfdfgdfgh"
    }
}
```

Participant Publish Error: このイベントは、参加者がステージへの公開の試行に失敗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-ZyXwvu1T2s",
        "event_name": "Participant Publish Error",
        "event_time": "2024-08-13T14:38:17.089061676Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "error_code": "BITRATE_EXCEEDED",
        "replica": true,
        "source_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij",
        "source_session_id": "st-sdfdfdfgdfgh"
    }
}
```

参加者のレプリケーション開始: このイベントは、参加者のレプリケーションが開始され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-ZyXwvu1T2s",
        "event_name": "Participant Replication Start",
        "event_time": "2025-11-18T16:40:32Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "destination_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/
XYZdef1G2hij",
        "destination_session_id":  "aBC1c2d3e4f"
    }
}
```

参加者のレプリケーション終了: このイベントは、参加者のレプリケーションが終了したときに送信されます。レプリケーションは、StopParticipantReplication API オペレーションによって終了することがあります。また、パブリッシャーが公開を停止した場合や、パブリッシャーが公開を停止し、再接続ウィンドウが期限切れになった場合にも終了することがあります。

```
{
    "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-ZyXwvu1T2s",
        "event_name": "Participant Replication End",
        "event_time": "2025-11-18T16:40:32Z",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f",
        "destination_stage_arn": "arn:aws:ivs:us-west-2:123456789012:stage/
XYZdef1G2hij",
        "destination_session_id":  "aBC1c2d3e4f"
    }
}
```

トークン交換済み: このイベントは、既存の参加者トークンが新しい参加者トークンと交換されたときに送信され、トークン機能のアップグレードまたはダウングレードおよびトークン属性の更新が行われます。

```
{
    "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-ZyXwvu1T2s",
        "event_name": "Token Exchanged",
        "event_time": "2025-11-12T20:54:53Z",
        "user_id": "UpdatedUser",
        "participant_id": "xYz1c2d3e4f",
        "previous_token": {
            "capabilities": ["SUBSCRIBE"],
            "attributes": {
                "role": "viewer"
            },
            "user_id": "InitialUser",
            "expiration_time": "2025-11-12T21:54:52Z"
        },
        "new_token": {
            "capabilities": ["SUBSCRIBE", "PUBLISH"],
```

```
            "attributes": {
                "role": "moderator"
            },
            "user_id": "UpdatedUser",
            "expiration_time": "2025-11-12T22:54:52Z"
        }
    }
}
```

IVS 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

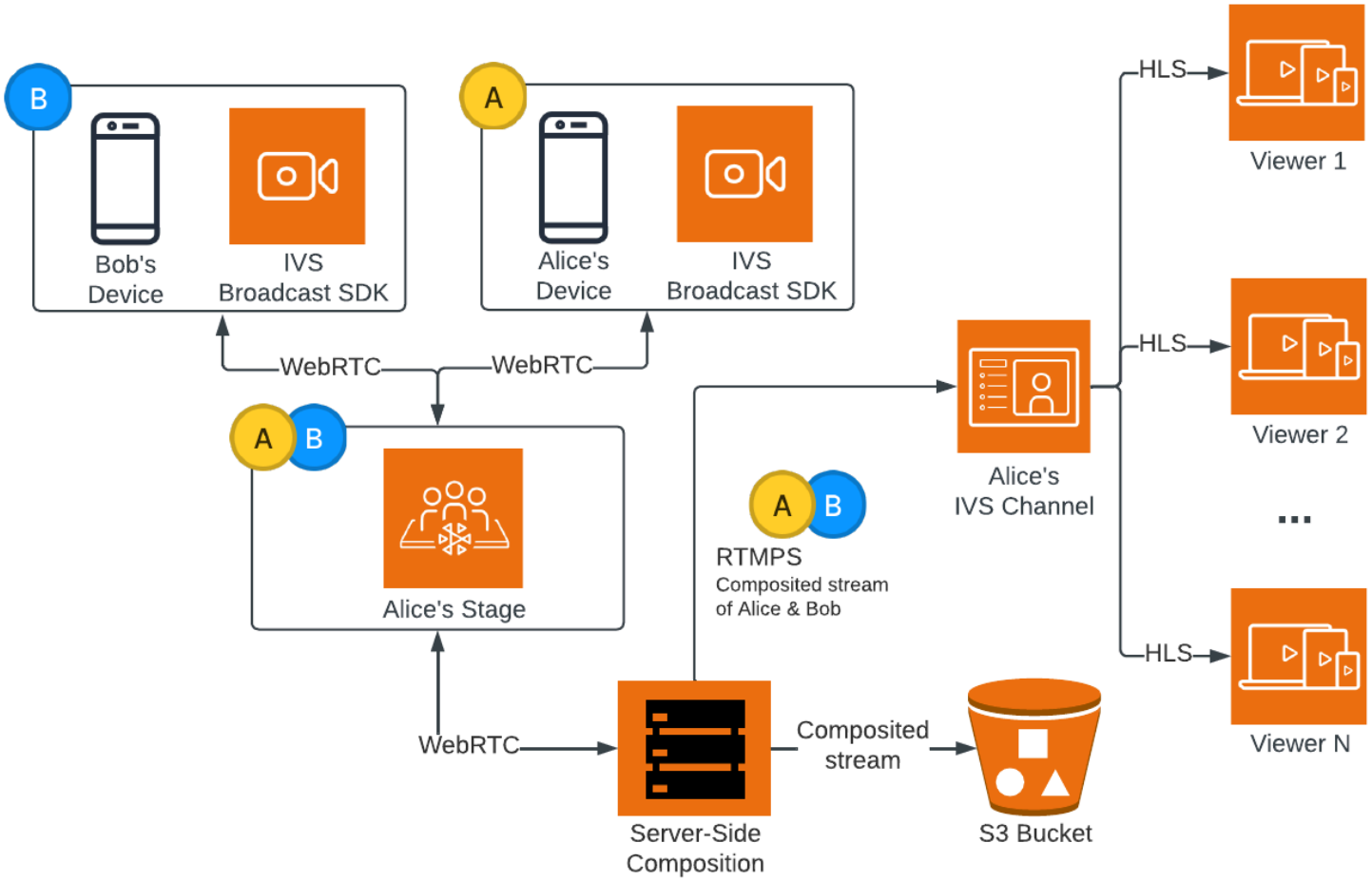
サーバーサイドコンポジションでは、IVS サーバーを使用してステージ参加者全員からの音声と動画を合成し、IVS チャネル (より多くの視聴者に配信する場合など) または S3 バケットに送信します。サーバーサイドコンポジションは、ステージのホームリージョンにある IVS コントロールプレーンオペレーションを介して呼び出されます。

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動画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。

トピック

サーバーサイドコンポジションの概要

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

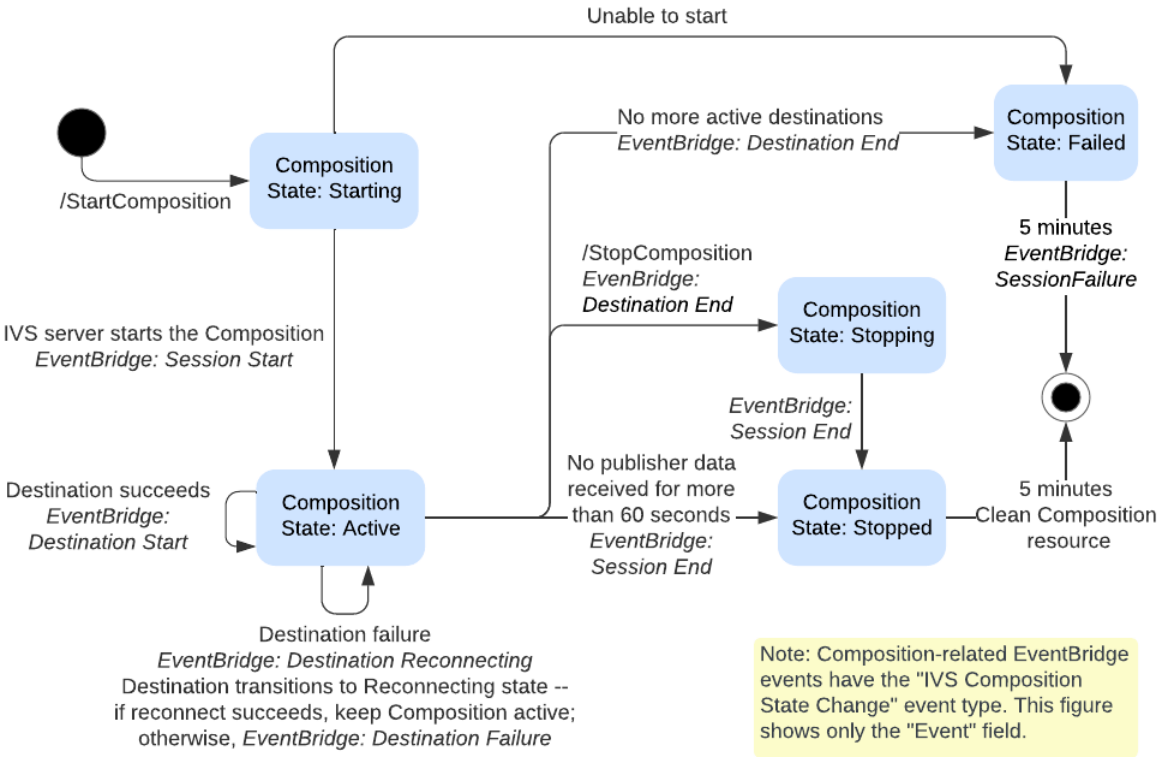
ていたりしても、サーバーはそれに適応するので、すべての視聴者にスムーズなストリームを提供できます。

- 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。

Composition のライフサイクル

下の図は、コンポジションの状態遷移を示します。



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. ユーザーが StartComposition オペレーションを呼び出したとき、コンポジションリソースが作成されます。
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 Real-Time Streaming 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。

- ユーザーがコンポジションを停止 – StopComposition オペレーションが呼び出された場合、IVS によってコンポジションの適切なシャットダウンが開始され、「Destination End」イベントの後に「Session End」イベントが送信されます。
- コンポジションが自動シャットダウンを実行 – IVS ステージが削除された、または IVS ステージにアクティブに配信する参加者がいない状態が 60 秒続いた場合、コンポジションが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。

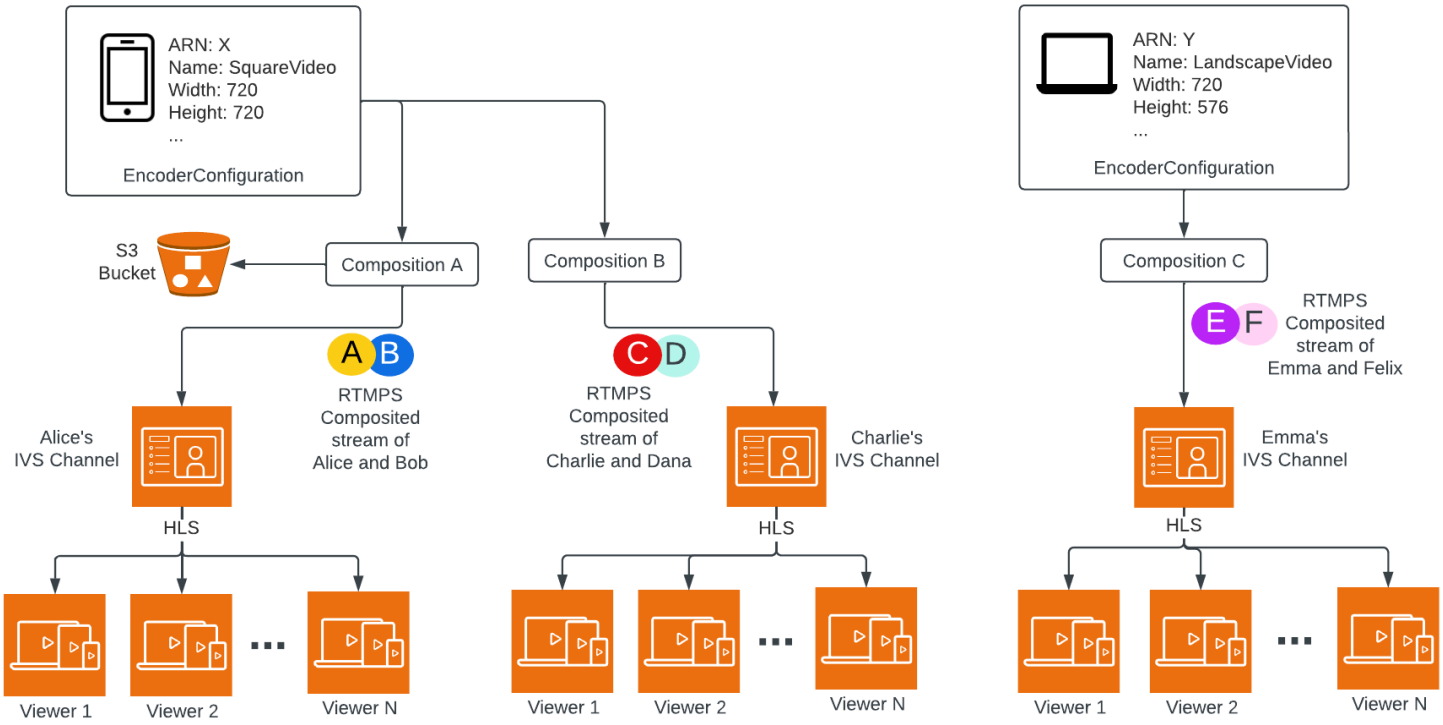
4. `STOPPED` あるいは `FAILED` 状態になったコンポジションは、その 5 分後に自動的にクリーンアップされます。(それ以降は、ListCompositions や GetComposition によって取得されなくなります)。

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。StartComposition オペレーションを呼び出すたびに、EncoderConfiguration を再利用できます。
- コンポジションオペレーションはビデオコンポジションを追跡し、IVS チャネルに出力します。
- 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 を作成して StartComposition オペレーションを呼び出すときに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 Real-Time Streaming API リファレンス」を参照してください。

Layouts

StartComposition オペレーションには、グリッドおよび PiP (ピクチャーインピクチャー) の 2 つのレイアウトオプションがあります。

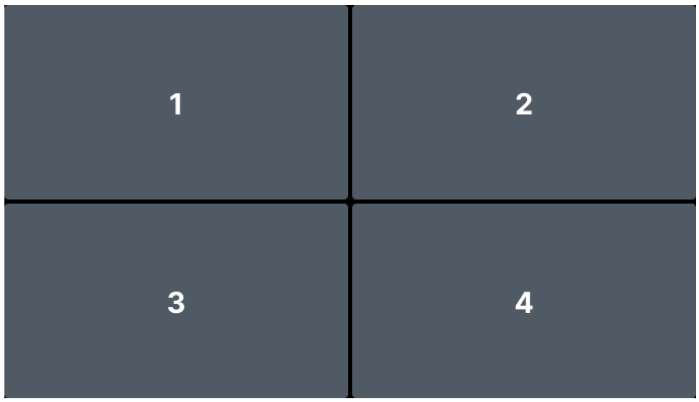
グリッドレイアウト

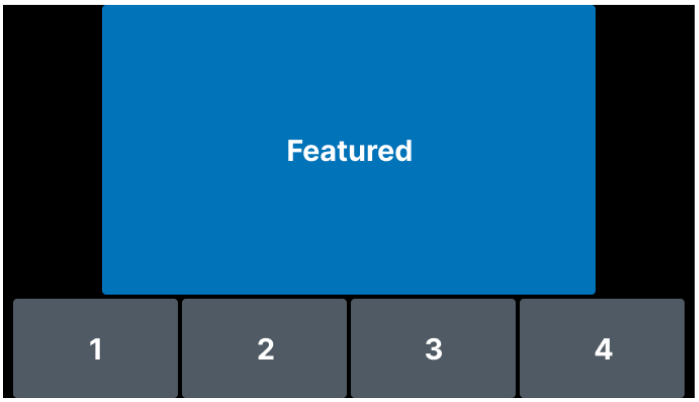
グリッドレイアウトは、ステージ参加者を等サイズのスロットのグリッドに配置します。カスタマイズ可能なプロパティがいくつか用意されています。

- `videoAspectRatio` は、ビデオタイルのアスペクト比を制御するように参加者表示モードを設定します。
- `videoFillMode` は、ビデオコンテンツが参加者タイルにどのように適合するかを定義します。
- `gridGap` は、参加者タイル間の間隔をピクセル単位で指定します。
- `omitStoppedVideo` では、停止したビデオストリームをコンポジションから除外できます。
- `featuredParticipantAttribute` は、注目のスロットを識別します。これを設定すると、注目の参加者はメイン画面の大きなスロットに表示され、他の参加者はその下に表示されます。
- `participantOrderAttribute` は、参加者トークンの属性値に基づきカスタム参加者の順序付けを有効にします。指定した場合、参加者は属性値で数値順に並べられ、属性がない参加者は到着

時刻の順序にフォールバックします。これにより、決定論的な配置がオプションで実施でき、ロールベースのレイアウトが可能になります。

グリッドレイアウト (すべてのフィールドの有効な値とデフォルトを含む) の詳細については、「GridConfiguration データ型」を参照してください。





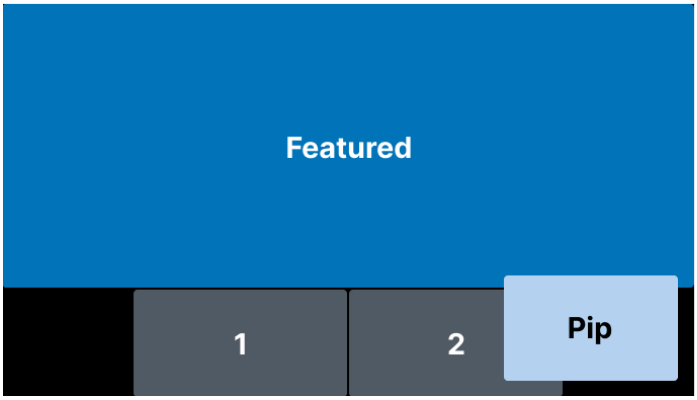
ピクチャインピクチャ (PiP) レイアウト

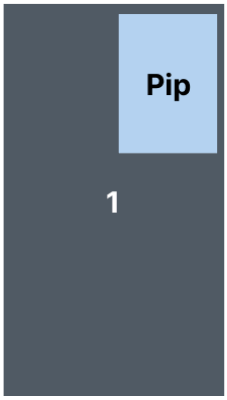
PiP レイアウトでは、参加者をオーバーレイウィンドウに表示することができ、そのサイズ、位置、動作を設定できます。これらのプロパティには、次のものがあります。

- `pipParticipantAttribute` は PiP ウィンドウの参加者を指定します。
- `pipPosition` は PiP ウィンドウのコーナー位置を決定します。
- `pipWidth` および `pipHeight` は、PiP ウィンドウの幅と高さを設定します。
- `pipOffset` は、PiP ウィンドウのオフセット位置を、最も近いエッジからのピクセル単位で設定します。
- `pipBehavior` は、他のすべての参加者が退出したときの PiP 動作を定義します。

グリッドレイアウトと同様に、PiP レイアウトは構成をさらにカスタマイズするための `featuredParticipantAttribute`、`omitStoppedVideo`、`videoFillMode`、`gridGap`、`participan` をサポートします。`participantOrderAttribute` は、PiP ウィンドウの参加者を選択する、参加者トークンの属性値に基づきグリッド参加者を配置するという両方のカスタム参加者の順序付けを有効にします。

PiP レイアウト (すべてのフィールドの有効な値とデフォルトを含む) の詳細については、「PipConfiguration データ型」を参照してください。





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。

重要: アプリケーションが、タイルのサイズや位置など、現在のレイアウトの特定の機能に依存していないことを確認してください。レイアウトの視覚的な改善は、いつでも導入できます。

IVS サーバーサイドコンポジションの開始方法

このドキュメントでは、サーバーサイドコンポジションの使用を開始するためのステップについて説明します。

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。

S3 バケットを作成するには、バケットの作成方法に関する S3 ドキュメントを参照してください。S3 バケットは IVS ステージと同じ AWS リージョンに作成する必要があります。

重要: 既存の S3 バケットを使用する場合:

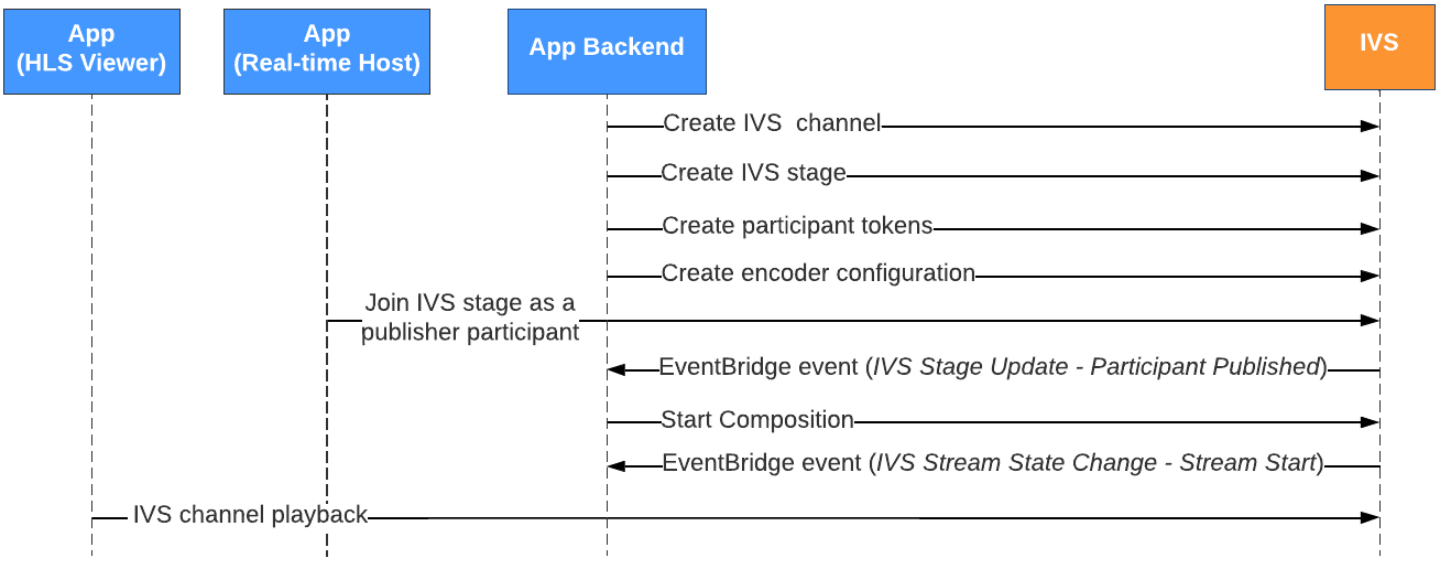
- [オブジェクト所有権] の設定は [バケット所有者に強制する] か、[バケット所有者を優先する] にする必要があります。
- [デフォルトの暗号化] 設定は、[Amazon S3 マネージドキーを用いたサーバー側の暗号化 (SSE-S3)] である必要があります。

詳細については、オブジェクトの所有権の制御と暗号化によるデータの保護に関する S3 ドキュメントを参照してください。

API の説明

以下では、参加者が発行した際に、ステージを IVS チャ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 Broadcast 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. 参加者が公開した EventBridge イベントを受信した場合は、希望のレイアウト設定で、StartComposition を呼び出します。
6. 数秒待ってから、チャ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。コンポジションオペレーションは `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 はコンポジションオペレーション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。

グリッドレイアウトの例

```
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"}}]' --layout '{"grid":
{"participantOrderAttribute":"order","featuredParticipantAttribute":"isFeatured","videoFillMode
```

PiP レイアウトの例

```
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 '{"pip":
{"participantOrderAttribute":"priority","pipParticipantAttribute":"isPip","pipOffset":10,"pipPo
```

注: このツールを使用して、レイアウトの選択に基づいて--layout 設定をより簡単に生成できます。

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 オペレーションまたは GetComposition オペレーション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。

カスタム参加者の順序付け

カスタム参加者の順序付けを使用すると、注目の参加者の配置や PiP ウィンドウの参加者の選択など、参加者トークンのカスタム属性値に基づいて、グリッドレイアウトと PiP レイアウトの両方で

参加者の配置を制御できます。これにより、決定論的な参加者配置が可能になり、ロールベースのレイアウトが有効になります。

カスタム順序の仕組み

レイアウト設定で participantOrderAttribute を指定すると、参加者は次のルールに従って順序付けられます。

- トークンに指定された順序付け属性を持つ参加者が最初に配置され、属性値で数値順にソートされます。
- 順序付け属性のない参加者は到着した順に後方に回され、順序付けされた参加者の後に配置されます。
- 複数の参加者が同じ順序値を持つ場合、ステージへの到着時間によってサブソートされます。
- 順序付けは数値ソート (辞書式ではない) を使用するため、「10」は「9」の後になります (「1」の後ではありません) 。
- 負の値はサポートされます。正の値の前に配置されます。
- 数値以外の値 (例 : 「abc」、「1.5」) は無効として扱われ、それらの参加者は到着順に後方に回されます。

重要: 参加者の順序付け (到着時間に基づくかカスタムに基づくかにかかわらず) は、コンポジションの開始後に有効になります。コンポジションが始まる前にステージに参加する参加者については、参加者の正しい順序は保証されません。

順序付け属性を使用したトークンの作成

カスタム参加者の順序付けを使用するには、参加者トークンの作成時に順序付け属性を含めます。

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

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=2

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=3
```

custom-participant-order 属性を、注目スロットと PiP ウィンドウの参加者を選択するための属性と組み合わせることができます。

```
aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=2,isFeatured=true

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=3,isFeatured=true

aws ivs-realtime create-participant-token --stage-arn "arn:aws:ivs:us-
east-1:123456789012:stage/u9OiE29bT7Xp" --attributes order=4,isPip=true
```

ユースケースの例

ユースケースの例を以下に示します。

- 一貫した配置 – 同じトークンで再接続する際、参加者の位置は維持されます。
- ロールベースの配置 – たとえば、order=1 の教師と order=2 の学生を指定できます。
- 優先度ベースのレイアウト – 低い順序の値を持つ VIP 参加者が最初に表示されます。
- 動的レイアウト – 複雑なシナリオでは、カスタム順序付けを `featuredParticipantAttribute` および `pipParticipantAttribute` と組み合わせることができます。
- クロスステージインタラクション – さまざまなステージのストリーマーがインタラクションを行う VS Mode 競技などのシナリオで参加者レプリケーションを使用する場合、順序付け属性を上書きして、宛先ステージコンポジションの配置を制御できます。

注: 参加者レプリケーションのユースケースでは、レプリケーションを開始するときに必要に応じて参加者属性 (順序属性を含む) を上書きして、レプリケート先ステージで目的のレイアウトを実現できます。

下位互換性

カスタム参加者の順序はオプション機能であり、完全な下位互換性が維持されます。`participantOrderAttribute` のない既存のコンポジションは、到着時刻の順序付けを使用して変更されずに動作し続けます。`participantOrderAttribute` が空の文字列に設定されている場合、システムはカスタム順序を完全に無視し、デフォルトの動作にフォールバックします。

IVS サーバーサイドコンポジションで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

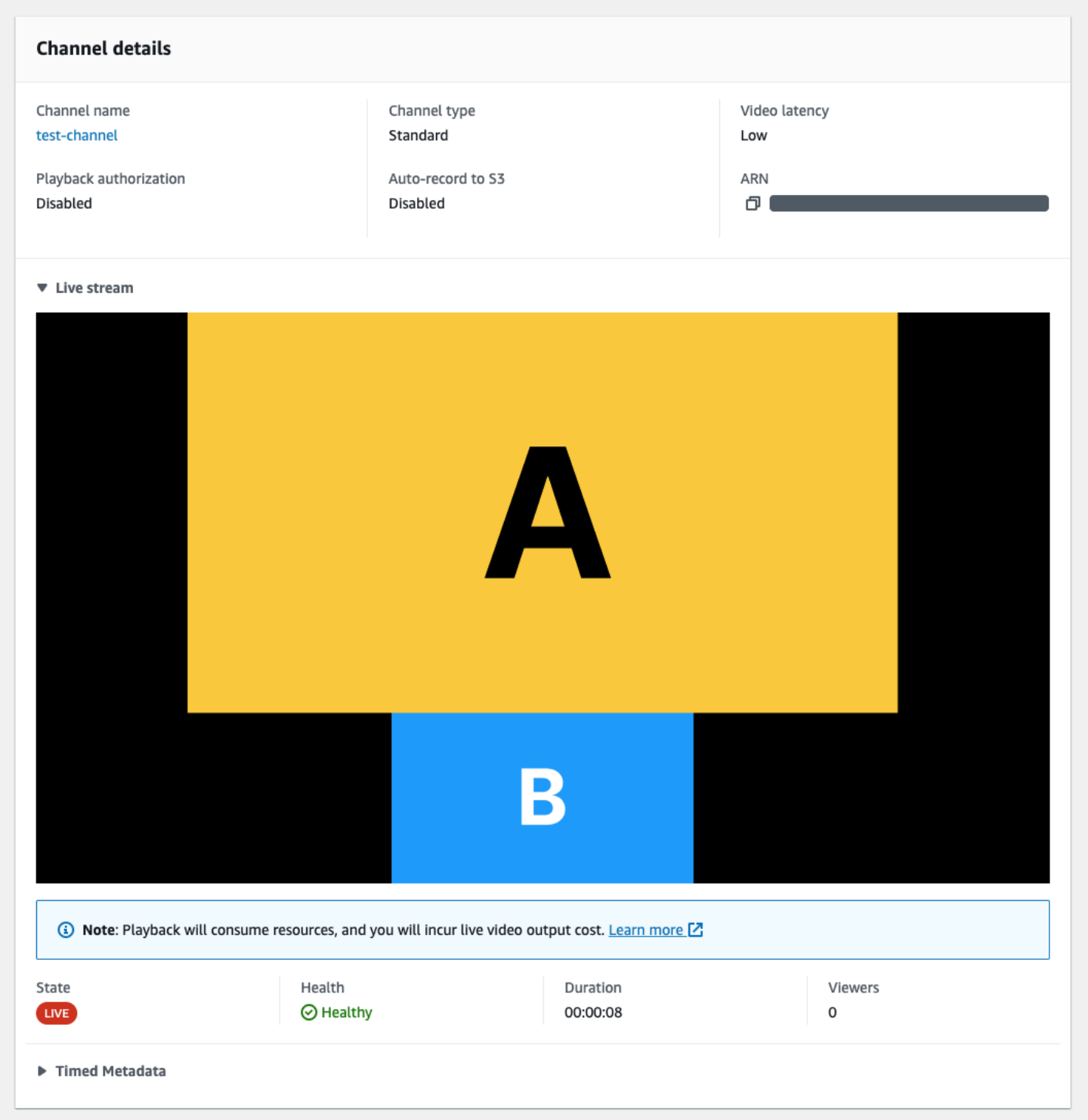
スクリーン共有機能によりコンポジションを開始するには、以下のコマンドを使用します。

```
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",
 "gridGap": 2,
 "omitStoppedVideo": false,
 "videoAspectRatio": "VIDEO"
 }
},
"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。

コンポジションをいつでも停止するには、StopComposition オペレーションを呼び出します。

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

既知の問題と回避策

このセクションでは、IVS サーバーサイドコンポジションを使用しているときに発生する可能性のある既知の問題をリストアップし、考えられる回避策を提案します。

- 一部のコンポジションで、無音状態が続いた後に音の途切れが短期間発生することがある。

  回避策: 該当なし

IVS 録画 | リアルタイムストリーミング

IVS Real-Time Streaming には 2 つの録画オプションがあります。

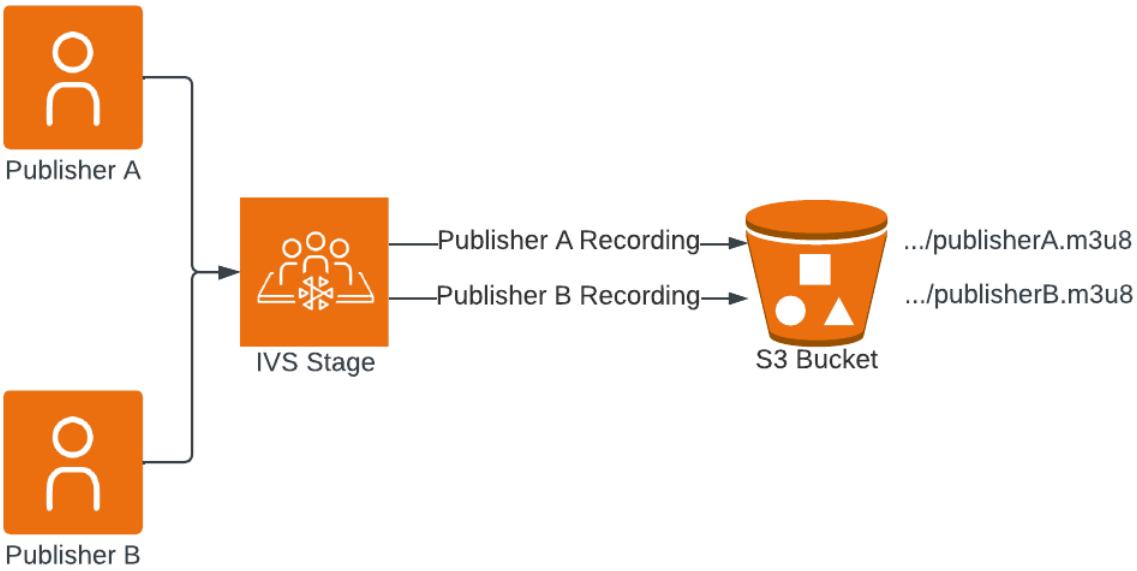
- 個々の参加者の録画では、各パブリッシャーのメディアは別々のファイルに記録されます。
- 対照的に、composite recording は、すべてのパブリッシャーのメディアを 1 つのビューに結合し、1 つのファイルに録画します。

個々の参加者の録画には Amazon IVS の追加料金はかかりませんが、composite recording には、エンコードされたビデオの時間単位の料金が発生します。どちらの録画オプションにも、標準の S3 ストレージとリクエストコストが発生します。詳細については、「Amazon IVS の料金」を参照してください。

よりカスタマイズ可能なソリューションについては、オープンソースの IVSStageSaver プロジェクトを独自のセルフホスト録画サービスの基盤として使用することを検討してください。

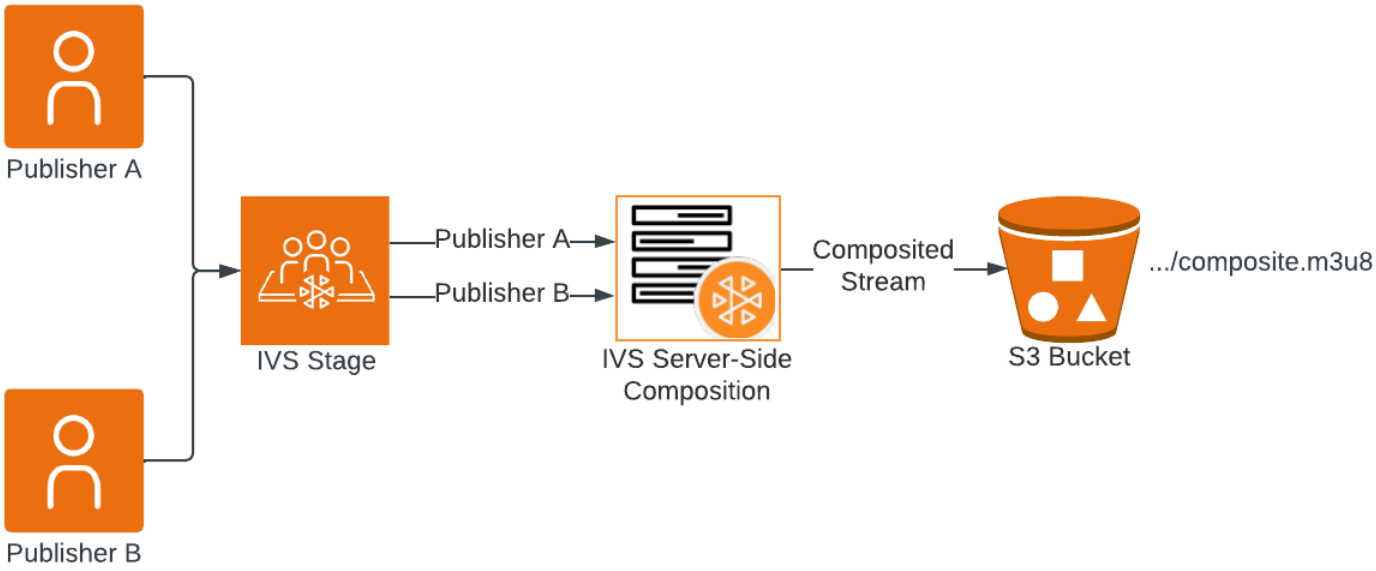
個々の参加者の録画

このオプションは、1 つのパブリッシャーを持つライブストリームや、特にモデレーション目的で、各パブリッシャーの個別の録画が必要な場合に最適です。詳細については、「個々の参加者の録画」を参照してください。



Composite Recording

このオプションは、複数のパブリッシャーのメディアを 1 つのビューに結合し、1 つのファイルに記録するため、ビデオオンデマンドエクスペリエンスに最適です。詳細については、「Composite Recording」を参照してください。



サムネイル

IVS リアルタイムストリーミングのサムネイル録画は、個々の参加者の録画と Composite Recording (複数の参加者) の両方に設定できます。サムネイル記録を有効または無効にしてサムネイルが生成される間隔を調整するには:

- 個々の参加者の録画の場合、 `thumbnailConfiguration` プロパティを使用します。
- Composite Recording の場合、`thumbnailConfigurations` プロパティを使用します。

サムネイルの間隔は 1～86400 秒 (24 時間) です。デフォルトでは、サムネイルの録画は無効になっています。詳細については、「Amazon IVS Real-Time Streaming API リファレンス」を参照してください。

サムネイル設定には `storage` フィールドが含まれています。これは、`SEQUENTIAL` および `LATEST` に設定できます。`storage` フィールドは、サムネイルの S3 ストレージ動作を決定します。

- `SEQUENTIAL` では、すべてのサムネイルが連続して保存されます。これがデフォルトです。
- `LATEST` では、最新のサムネイルのみが保存され、前のサムネイルが上書きされます。

SEQUENTIAL と LATEST の両方を指定すると、サムネイルは 2 つの個別の S3 パス (シーケンシャルアーカイブ用と最新のサムネイル用) に書き込まれます。

IVS 個々の参加者の録画 | リアルタイムストリーミング

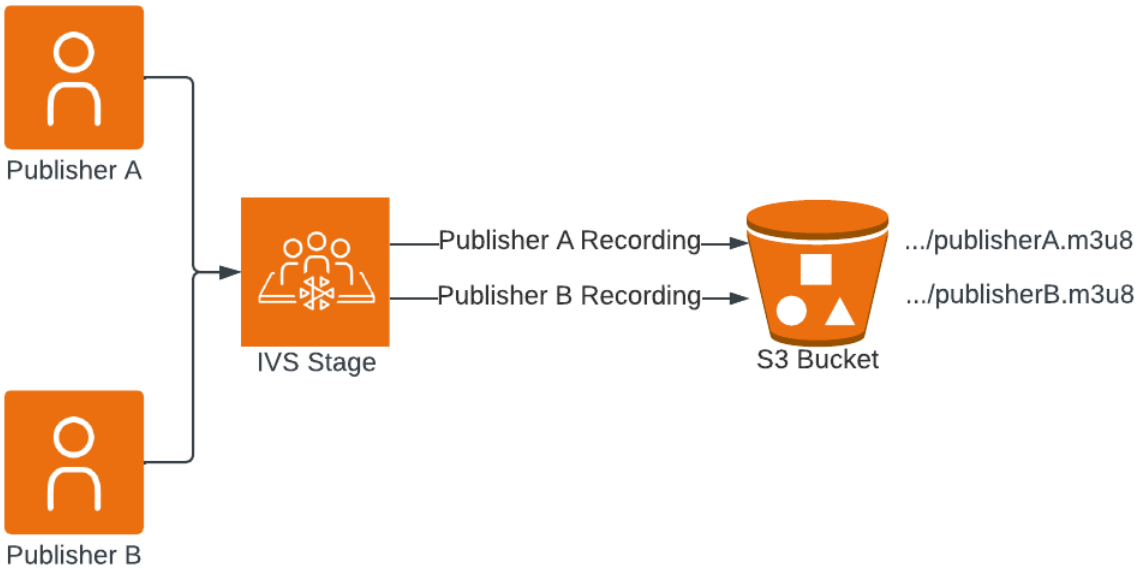
このドキュメントでは、IVS Real-Time Streaming で個々の参加者の録画を使用する方法について説明します。

標準の S3 ストレージとリクエストのコストが適用されます。サムネイルでは追加の IVS 料金は発生しません。詳細については、「Amazon IVS の料金」を参照してください。

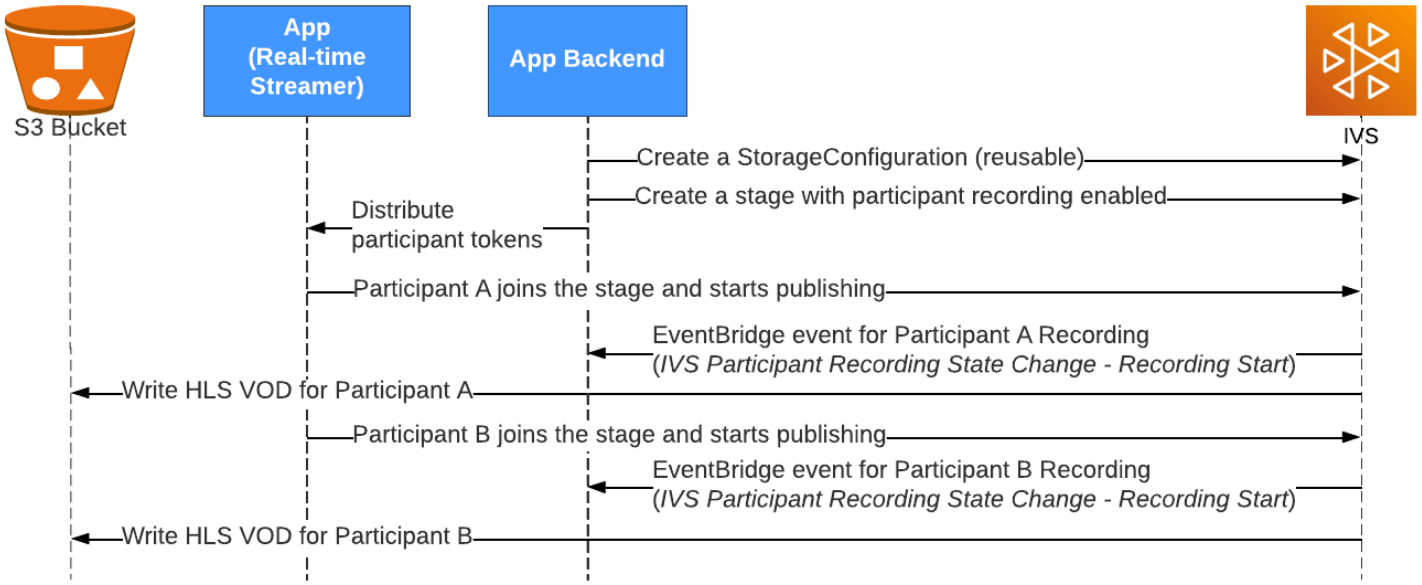
序章

個々の参加者の録画により、IVS Real-Time Streaming のお客様は IVS ステージパブリッシャーを個別に S3 バケットに録画できます。ステージで個々の参加者の録画が有効になっている場合、パブリッシャーコンテンツは、ステージへの公開が開始されると録画されます。

注: すべてのステージ参加者を 1 つのビデオに混在させる必要がある場合は、composite recording 機能が適しています。IVS リアルタイムストリーミングコンテンツの録画の概要については、「録画」を参照してください。



ワークフロー



1. S3 バケットの作成

VOD を書き込むには S3 バケットが必要です。詳細については、バケットの作成方法に関する S3 ドキュメントを参照してください。個々の参加者の録画には、IVS ステージと同じ AWS リージョンに S3 バケットを作成する必要があります。

重要: 既存の S3 バケットを使用する場合:

- [オブジェクト所有権] の設定は [バケット所有者に強制する] か、[バケット所有者を優先する] にする必要があります。
- [デフォルトの暗号化] 設定は、[Amazon S3 マネージドキーを用いたサーバー側の暗号化 (SSE-S3)] である必要があります。

詳細については、オブジェクトの所有権の制御と暗号化によるデータの保護に関する S3 ドキュメントを参照してください。

2. StorageConfiguration オブジェクトを作成する

バケットを作成したら、IVS Real-Time Streaming API を呼び出して StorageConfiguration オブジェクトを作成します。ストレージ設定が正常に作成されると、IVS は提供された S3 バケットに書き込むアクセス許可を持ちます。この StorageConfiguration オブジェクトは、複数のステージで再利用できます。

3. 属性を使用してステージ参加者トークンを作成します。

次に、個々の参加者の録画を有効にして (AutoParticipantRecordingConfiguration オブジェクトを設定して)、各パブリッシャーの参加者トークンと IVS ステージを作成する必要があります。

以下のリクエストは、2 つの参加者トークンと個々の参加者記録が有効になっているステージを作成します。

```
POST /CreateStage HTTP/1.1
Content-type: application/json

{
   "autoParticipantRecordingConfiguration": {
      "mediaTypes": ["AUDIO_VIDEO"],
      "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/AbCdef1G2hij",
      "thumbnailConfiguration": {
         "recordingMode": "INTERVAL",
         "storage": ["LATEST", "SEQUENTIAL"],
         "targetIntervalSeconds": 60
      }
   },
   "name": "TestStage",
   "participantTokenConfigurations": [
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "1"
      },
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "2"
      }
   ]
}
```

4. ステージにアクティブパブリッシャーとして参加させる

参加者トークンをパブリッシャーに配布し、ステージに参加させ、パブリッシュを開始します。

ステージに参加し、IVS Real-Time Streaming Broadcast SDK のいずれかを使用してステージへの公開を開始すると、参加者録画プロセスが自動的に開始され、録画が開始されたことを示す EventBridge イベントが送信されます。(イベントは IVS 参加者の録画状態の変更 - 録画開始。) 同時に、参加者録画プロセスは、設定された S3 バケットへの VOD およびメタデータファイルの書き込みを開始します。注: 非常に短い期間 (5 秒未満) で接続された参加者は、録画される保証はありません。

各録画の S3 プレフィックスを取得するには、次の 2 つの方法があります。

- EventBridge イベントを再生します。

```
{
   "version": "0",
   "id": "12345678-1a23-4567-a1bc-1a2b34567890",
   "detail-type": "IVS Participant Recording State Change",
   "source": "aws.ivs",
   "account": "123456789012",
   "time": "2024-03-13T22:19:04Z",
   "region": "us-east-1",
   "resources": ["arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"],
   "detail": {
      "session_id": "st-ZyXwvu1T2s",
      "event_name": "Recording Start",
      "participant_id": "xYz1c2d3e4f",
      "recording_s3_bucket_name": "ivs-recordings",
      "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z"
   }
}
```

- GetParticipant API オペレーションの使用 — レスポンスには、参加者が録画されている S3 バケットとプレフィックスが含まれます。リクエストは次のとおりです。

```
POST /GetParticipant HTTP/1.1
Content-type: application/json
{
   "participantID": "xYz1c2d3e4f",
   "sessionId": "st-ZyXwvu1T2s",
   "stageArn": "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
}
```

そして、次のレスポンスがあります。

```
Content-type: application/json
{
   "participant": {
      ...
      "recordingS3BucketName": "ivs-recordings",
      "recordingS3Prefix": "<stage_id>/<session_id>/<participant_id>",
      "recordingState": "ACTIVE",
      ...
   }
}
```

5. VOD を再生する

録画が確定したら、IVS プレイヤーを使用して録画を視聴できます。VOD 再生用に CloudFront ディストリビューションを設定する手順については、プライベートバケットからの録画コンテンツの再生を参照してください。

音声のみの録音

個々の参加者の録画/録音を設定する場合、S3 バケットに書き込むよう選択できるのはオーディオ HLS セグメントのみです。この機能を使用するには、ステージの作成時に AUDIO_ONLY mediaType を選択します。

```
POST /CreateStage HTTP/1.1
Content-type: application/json

{
   "autoParticipantRecordingConfiguration": {
      "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/AbCdef1G2hij",
      "mediaTypes": ["AUDIO_ONLY"],
      "thumbnailConfiguration": {
         "recordingMode": "DISABLED"
      }
   },
   "name": "TestStage",
   "participantTokenConfigurations": [
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
```

```
         "userId": "1"
      },
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "2"
      }
   ]
}
```

サムネイルのみの録画

個々の参加者の録画を設定する場合、S3 バケットに書き込むよう選択できるのはサムネイルのみです。この機能を使用するには、ステージの作成時に mediaType を NONE に設定します。これで HLS セグメントが生成されなくなります。サムネイルは引き続き作成され、S3 バケットに書き込まれます。

```
POST /CreateStage HTTP/1.1
Content-type: application/json
{
   "autoParticipantRecordingConfiguration": {
      "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-
configuration/AbCdef1G2hij",
      "mediaTypes": ["NONE"],
      "thumbnailConfiguration": {
         "recordingMode": "INTERVAL",
         "storage": ["LATEST", "SEQUENTIAL"],
         "targetIntervalSeconds": 60
      }
   },
   "name": "TestStage",
   "participantTokenConfigurations": [
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "1"
      },
      {
         "capabilities": ["PUBLISH", "SUBSCRIBE"],
         "duration": 20160,
         "userId": "2"
      }
   ]
```

```
}
```

録画の内容

個々の参加者の録画がアクティブになると、ステージの作成時に提供された S3 バケットに HLS ビデオセグメント、メタデータファイル、サムネイルの書き込みが開始します。このコンテンツは、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。

録画が確定すると、IVS 参加者の録画状態が変更される (録画終了イベントが EventBridge を介して送信される) ことに注意してください。録画したストリームの再生または処理は、必ず Recording End イベントの送信後に行うことをお勧めします。詳細については、「IVS Real-Time Streaming で Amazon EventBridge を使用する」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。

```
s3://mybucket/stageId/stageSessionId/participantId/timestamp
   events
      recording-started.json
      recording-ended.json
   media
      hls
  multivariant.m3u8
         high
            playlist.m3u8
            1.mp4
      thumbnails
         high
            1.jpg
            2.jpg
      latest_thumbnail
         high
            thumb.jpg
```

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、録画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 プレイヤーで再生できます。設定されている場合、 thumbnails と latest_thumbnail のサブフォルダには、録画セッション中に生成された JPEG サムネイルメディアファイルが含まれます。

フラグメント化された個々の参加者の記録をマージする

記録設定の recordingReconnectWindowSeconds プロパティを使用すると、ステージパブリッシャーがステージから切断してから再接続した場合に IVS が前のセッションと同じ S3 プレフィックスに記録を試みる際、時間枠 (秒単位) を指定できます。つまり、パブリッシャーが切断してから指定された時間内で再接続した場合、複数の記録は 1 つの記録として見なされてマージされます。

サムネイル記録が SEQUENTIAL モードで有効になっている場合、サムネイルも同じ recordingS3Prefix の下でマージされます。記録がマージされると、サムネイルカウンターは前の記録用に書き込まれた前のサムネイル値から再開始されます。

Amazon EventBridge の IVS 記録状態変更イベント: IVS が新しいストリームが開始されないことを確認するために待機するため、Recording End イベントおよび recording-ended JSON メタデータファイルは最低 recordingReconnectWindowSeconds 遅延されます。

マージストリーム機能の設定手順については、「Amazon IVS Real-Time Streaming の開始方法」の「ステップ 2: ステージを作成する」を参照してください。

対象

同じ S3 プレフィックスを使用して複数の記録をマージするには、すべての記録が特定の条件を満たす必要があります。

- ステージの AutoParticipantRecordingConfiguration の recordingReconnectWindowSeconds プロパティにおける値は 0 より大きく設定されます。
- VOD アーティファクトの書き込みに使用される StorageConfigurationArn は各記録において同様です。
- 参加者がステージを離れてから再参加するまでの秒単位の時間差は recordingReconnectWindowSeconds 以下です。

recordingReconnectWindowSeconds のデフォルト値は 0 であり、マージが無効になることに注意してください。

複数の参加者の記録を同期させる

個々の参加者の録画には、HLS プレイリストの EXT-X-PROGRAM-DATE-TIME タグが含まれます。これにより、後処理中に複数の参加者の記録を同期するための正確な UTC タイムスタンプをミリ秒の精度で提供します。

複数の参加者を個別に録画し、同期されたコンポジション (サイドバイサイドレイアウトやピクチャーインピクチャーレイアウトなど) を作成する場合は、参加者がさまざまなタイミングでステージに参加したり、ネットワークの中断によって不連続になったりした場合でも、これらのタイムスタンプを使用して録画を正確に調整できます。

各参加者の HLS プレイリストには、以下を示す EXT-X-PROGRAM-DATE-TIME タグが含まれています。

- 録画の開始 (最初のセグメント) 。
- ステッチングが発生したときなど、録画中の不連続ポイント。

これらのタイムスタンプは、ミリ秒単位の精度を持ち、同じタイムリファレンスを使用してすべての参加者間で同期されます。

HLS プレイリストの例

```
#EXTM3U
#EXT-X-VERSION:7
#EXT-X-TARGETDURATION:12
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-MAP:URI="init-0.mp4"
#EXT-X-PROGRAM-DATE-TIME:2024-01-01T12:00:00.000Z
#EXTINF:3.30091,
0.mp4
#EXTINF:5.63794,
1.mp4
#EXTINF:2.74290,
2.mp4
#EXT-X-DISCONTINUITY
#EXT-X-MAP:URI="init-1.mp4"
#EXT-X-PROGRAM-DATE-TIME:2024-01-01T12:00:52.772Z
#EXTINF:2.54412,
```

```
3.mp4
#EXTINF:5.63649,
4.mp4
```

EXT-X-PROGRAM-DATE-TIME タグは、最初のセグメントと各不連続ポイントの正確な UTC 時間を提供し、他の参加者の録画との正確な同期を可能にします。

同期ワークフロー

複数の参加者の録画を同期させるには、各参加者の HLS プレイリストから EXT-X-PROGRAM-DATE-TIME タイムスタンプを抽出し、それを使用してタイムオフセットを計算します。これらのオフセットは、FFmpeg などのビデオ処理ツールを使用して後処理コンポジション中に適用できます。録画に途切れが発生している場合でも、その時点のタイムスタンプが正確な同期を保つためのタイミング基準を提供します。これにより、録画全体を通じて正確な同期が維持されます。

注: 後処理なしでリアルタイムに同期された出力を得たい場合は、参加者ごとの個別録画ではなくサーバーサイドコンポジションの利用を検討してください。

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	録画のソースとして使用されているステージの ARN。
session_id	string	はい	参加者が録画されるステージの session_id を表す文字列。
participant_id	string	はい	録画された参加者の識別子を表す文字列。
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。また、recording_ended_at につい

フィールド	Type	必須	説明
			ては、以下の注を参照してください。
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。 注: recording_started_at および 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_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。

フィールド	Type	必須	説明
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。
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。

フィールド	Type	必須	説明
thumbnails	オブジェクト	条件付き	サムネイル出力を記述する列挙型フィールド。これを使用できるのは、サムネイル設定の storage フィールドに SEQUENTIAL が含まれる場合のみです。
path	string	はい	シーケンシャルサムネイル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。
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 (サムネイルバリアント) 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サムネイル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。
latest_thumbnail	オブジェクト	条件付き	サムネイル出力を記述する列挙型フィールド。これを使用できるのは、サムネイル設定の storage フィールドに LATEST が含まれる場合のみです。
path	string	はい	latest_thumbnail が格納されている S3 プレフィックスからの相対パス。
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 (サムネイルバリアント) の配列。レンディションは必ず 1 つ以上。

フィールド	Type	必須	説明
path	文字列	はい	このレンディションの最新のサムネイルが格納されている S3 プレフィックスからの相対パス。
version	string	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:us-west-2:aws_account_id:stage/AbCdef1G2hij",
   "session_id": "st-ZyXwvu1T2s",
   "participant_id": "xYz1c2d3e4f",
   "recording_started_at": "2024-03-13T13:17:17Z",
   "recording_status": "RECORDING_STARTED",
   "media": {
      "hls": {
         "path": "media/hls",
         "playlist": "multivariant.m3u8",
         "renditions": [
            {
               "path": "high",
               "playlist": "playlist.m3u8"
            }
         ]
      },
      "thumbnails": {
         "path": "media/thumbnails",
         "renditions": [
            {
               "path": "high"
            }
         ]
      },
      "latest_thumbnail": {
         "path": "media/latest_thumbnail",
         "renditions": [
            {
```

```
                "path": "high"
            }
         ]
      }
   }
}
```

例: recording-ended.json

```
{
   "version": "v1",
   "stage_arn": "arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij",
   "session_id": "st-ZyXwvu1T2s",
   "participant_id": "xYz1c2d3e4f",
   "recording_started_at": "2024-03-13T19:44:19Z",
   "recording_ended_at": "2024-03-13T19:55:04Z",
   "recording_status": "RECORDING_ENDED",
   "media": {
      "hls": {
         "duration_ms": 645237,
         "path": "media/hls",
         "playlist": "multivariant.m3u8",
         "renditions": [
            {
               "path": "high",
               "playlist": "playlist.m3u8"
            }
         ]
      },
      "thumbnails": {
         "path": "media/thumbnails",
         "renditions": [
            {
               "path": "high"
            }
         ]
      },
      "latest_thumbnail": {
         "path": "media/latest_thumbnail",
         "renditions": [
            {
               "path": "high"
            }
```

```
         ]
      }
   }
}
```

例: recording-failed.json

```
{
   "version": "v1",
   "stage_arn": "arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij",
   "session_id": "st-ZyXwvu1T2s",
   "participant_id": "xYz1c2d3e4f",
   "recording_started_at": "2024-03-13T19:44:19Z",
   "recording_ended_at": "2024-03-13T19:55:04Z",
   "recording_status": "RECORDING_ENDED_WITH_FAILURE",
   "media": {
      "hls": {
         "duration_ms": 645237,
         "path": "media/hls",
         "playlist": "multivariant.m3u8",
         "renditions": [
            {
               "path": "high",
               "playlist": "playlist.m3u8"
            }
         ]
      },
      "thumbnails": {
         "path": "media/thumbnails",
         "renditions": [
            {
               "path": "high"
            }
         ]
      },
      "latest_thumbnail": {
         "path": "media/latest_thumbnail",
         "renditions": [
            {
               "path": "high"
            }
         ]
      }
```

```
    }
}
```

記録を MP4 に変換する

個々の参加者の記録は、プレイリストとフラグメント化された MP4 (fMP4) セグメントで構成される HLS 形式で保存されます。HLS 記録を単一の MP4 ファイルに変換するには、FFmpeg をインストールして次のコマンドを実行します。

```
ffmpeg -i /path/to/playlist.m3u8 -i /path/to/playlist.m3u8 -map 0:v -map 1:a -c copy
 output.mp4
```

IVS 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。

標準の S3 ストレージとリクエストのコストが適用されます。サムネイルでは追加の IVS 料金は発生しません。詳細については、「Amazon IVS の料金」を参照してください。

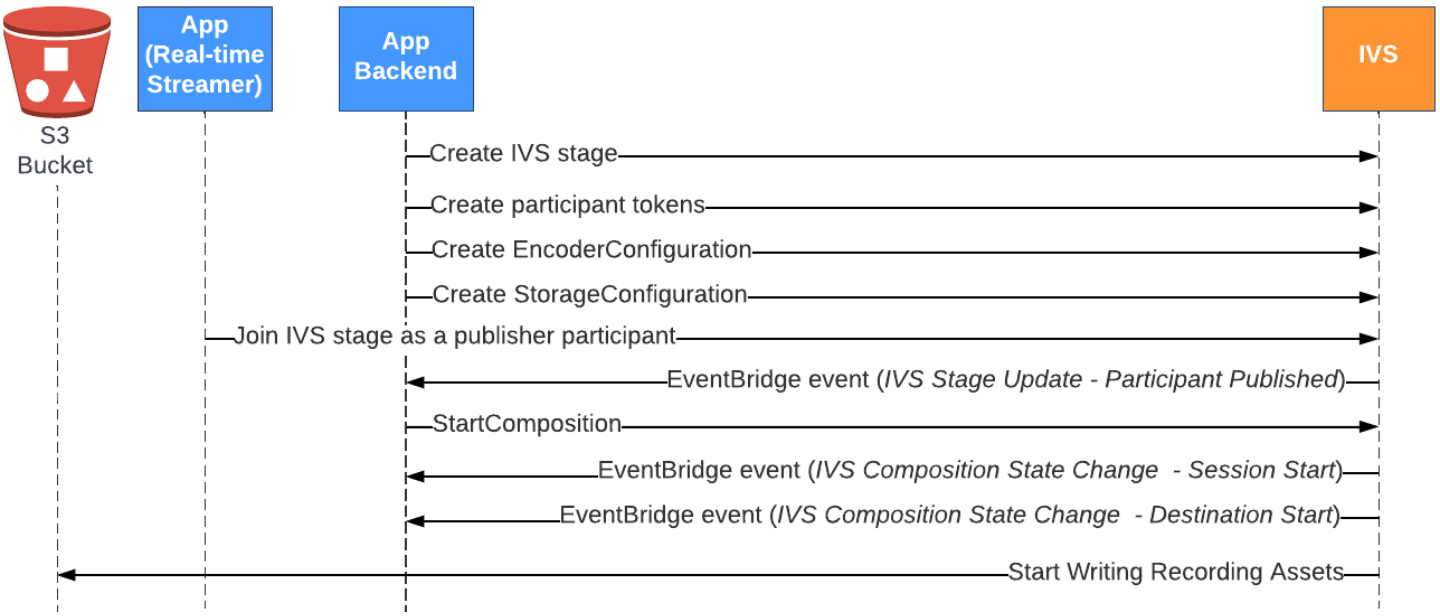
前提条件

Composite Recording 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。以下では、EventBridge イベントを使用してコンポジションを S3 バケットに録画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. EncoderConfiguration (録画したビデオのレンダリング方法を表すオブジェクト) を作成します。
3. S3 バケットと StorageConfiguration (録画内容を保存する場所) を作成します。

   重要: 既存の S3 バケットを使用する場合、オブジェクト所有権設定は [バケット所有者に強制する] か、[、バケット所有者を優先する] である必要があります。詳細については、オブジェクトの所有権の制御に関する S3 ドキュメントを参照してください。
4. ステージに参加して公開します。

5. 参加者が公開した EventBridge イベントを受信したら、S3 DestinationConfiguration オブジェクトを送信先として StartComposition 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「コンポジションのライフサイクル」を参照してください。

Composite Recording の例: S3 バケットの送信先での StartComposition

以下の例では、S3 をコンポジションの唯一の送信先として指定する StartComposition オペレーション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 Real-Time Streaming 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",
     "thumbnailConfigurations": [
        {
    "storage": ["LATEST", "SEQUENTIAL"],
    "targetIntervalSeconds": 30
               }
     ]
  }
      }
   ],
   "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",
```

```
                "thumbnailConfigurations": [
                    {
                "storage": ["LATEST", "SEQUENTIAL"],
                "targetIntervalSeconds": 30
                            }
                ]
                    }
                },
                "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 レスポンスにある `recordingPrefix` フィールドを使用して、録画の内容を保存する場所を決定できます。

録画の内容

コンポジションが `ACTIVE` 状態に遷移すると、StartComposition の呼び出し中に指定された S3 バケットへの HLS ビデオセグメント、メタデータファイル、サムネイル (設定されている場合) の書き込みが開始されます。このコンテンツは、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」イベントが発生すると、マニフェストファイル、ビデオセグメント、サムネイル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 Real-Time Streaming で Amazon EventBridge を使用する」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
      thumbnails
      latest_thumbnail
```

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。設定されている場合、 thumbnails と latest_thumbnail のサブフォルダには、コンポジションセッション中に生成された JPEG サムネイルメディアファイルが含まれます。

StorageConfiguration のバケットポリシー

StorageConfiguration オブジェクトが作成されると、IVS は指定した 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。

JSON

```
{
    "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

フィールド	Type	必須	説明
			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。
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。

フィールド	Type	必須	説明
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。
thumbnails	オブジェクト	条件付き	サムネイル出力を記述する列挙型フィールド。これを使用できるのは、サムネイル設定の storage フィールドに SEQUENTIAL が含まれる場合のみです。
path	string	はい	シーケンシャルサムネイル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。
resolutions	オブジェクト	はい	メタデータオブジェクトの解像度 (サムネイルバリアント) の配列。常に少なくとも 1 つの解像度があります。
path	string	はい	この解像度のサムネイル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。
resolution_height	int	はい	サムネイルのピクセル解像度の高さ。
resolution_width	int	はい	サムネイルのピクセル解像度の幅。

フィールド	Type	必須	説明
latest_thumbnail	オブジェクト	条件付き	サムネイル出力を記述する列挙型フィールド。これを使用できるのは、サムネイル設定の storage フィールドに LATEST が含まれる場合のみです。
path	string	はい	latest_thumbnail が格納されている S3 プレフィックスからの相対パス。
resolutions	オブジェクト	はい	メタデータオブジェクトの解像度 (サムネイルバリアント) の配列。常に少なくとも 1 つの解像度があります。
path	string	はい	この解像度の最新のサムネイルが保存されている S3 プレフィックスからの相対パス。
resolution_height	int	はい	最新のサムネイルのピクセル解像度の高さ。
resolution_width	int	はい	最新のサムネイルのピクセル解像度の幅。

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
        }
```

```
      ]
    },
    "thumbnails": {
      "path": "media/thumbnails",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "latest_thumbnail": {
      "path": "media/latest_thumbnail",
      "resolutions": [
        {
          "path": "1280x720",
          "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
        }
      ]
    },
    "thumbnails": {
      "path": "media/thumbnails",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "latest_thumbnail": {
      "path": "media/latest_thumbnail",
      "resolutions": [
        {
          "path": "1280x720",
          "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
        }
      ]
    },
    "thumbnails": {
      "path": "media/thumbnails",
      "resolutions": [
        {
          "path": "1280x720",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    },
    "latest_thumbnail": {
      "path": "media/latest_thumbnail",
      "resolutions": [
        {
          "path": "1280x720",
          "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、Amazon CloudFront CDN (コンテンツ配信ネットワーク) を使用してこれらのファイルを再生することができます。

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 CloudFront 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。オリジンアクセスコントロール (OAC) を作成すると、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 は特別な CloudFront ユーザーで、プライベートオリジンバケットに対する読み取りアクセス許可を持ちます。ディストリビューションの作成後、CloudFront コンソールまたは API を使用して OAC を作成することができます。「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 を有効にした CloudFront 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 を有効にした CloudFront ディストリビューションを設定し、どのドメインからでも録画を再生できるようにする方法を説明します。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

「CloudFront 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。

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 バケットに設定した StorageConfiguration 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になっていることを確認し、バケットポリシーをコピーします ([ポリシーをコピー] 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 ポリシーのみが含まれているはずです。
4. S3 バケットを指定し、StorageConfiguration 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 がコンテンツを読み取ることを許可し、もう 1 つは IVS がコンテンツを書き込むことを許可します。CloudFront と IVS アクセスを含む最終的なバケットポリシーの例を「例: CloudFront と IVS アクセスを含む S3 バケットポリシー」に示します。

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、バケットポリシーを更新したら、IVS プレーヤーを使用して録画を再生できるはずです。

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 を実行すると、CloudFront URL を介してビデオファイルを使用できるようになります。CloudFront の URL は、Amazon CloudFront コンソールの [詳細] タブにある [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. CloudFront ディストリビューションを介して録画した動画を再生するには、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. CloudFront URL の末尾にオブジェクトキーを追加します。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 アクセスを含む S3 バケットポリシー

以下のスニペットは、CloudFront がプライベートバケットにコンテンツを読み取り、IVS 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 StorageConfiguration に関連する ID が含まれている必要があります。

JSON

```
{
  "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

- 構成が S3 バケットに書き込まれません – 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てください。また、バケットポリシーをチェックして IVS がバケットにアクセスできることを確認してください。「StorageConfiguration のバケットポリシー」を参照してください。
- ListCompositions を実行してもコンポジションが見つかりません – コンポジションは一時的なリソースです。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。

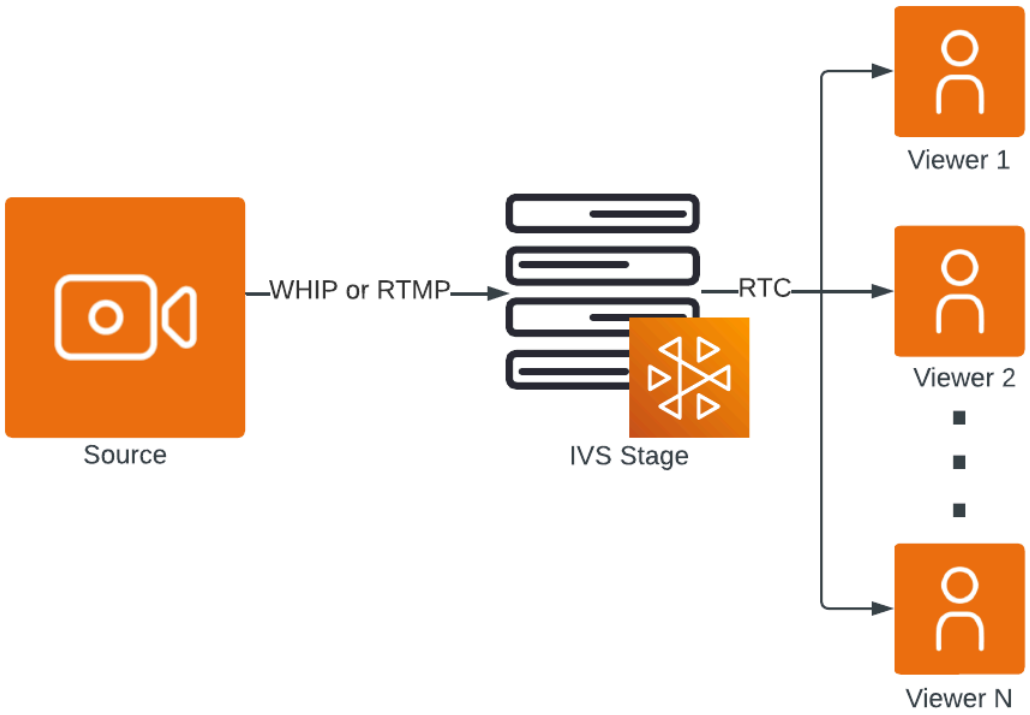
既知の問題

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。

IVS ストリーム取り込み | リアルタイムストリーミング

IVS Broadcast SDK を使用する代わりに、WHIP または RTMP ソースから IVS ステージにビデオを公開できます。このアプローチは、OBS Studio やハードウェアエンコーダーからビデオを公開する場合など、SDK の使用が実行不可能または推奨されないワークフローに柔軟性を提供します。サードパーティーソリューションと IVS のパフォーマンスや互換性を保証することはできないため、可能な限り IVS Broadcast SDK を使用することをお勧めします。

この図は、WHIP と RTMP を使用した公開の仕組みを示しています。



サポートされるプロトコル

IVS Real-Time Streaming は、いくつかの取り込みプロトコルをサポートしています。

- RTMP と RTMPS – RTMP (Real-Time Messaging Protocol) はネットワーク上でビデオを送信するための業界標準です。RTMPS は TLS 経由で動作する RTMP の安全なバージョンです。

  IVS は、E-RTMP (拡張 RTMP) のマルチトラックビデオ機能をサポートします。IVS RTMP Publishing ドキュメントの「E-RTMP Multitrack Video」を参照してください。
- WHIP (WebRTC-HTTP Ingestion Protocol) — WebRTC 取り込みを標準化するために開発された IETF ドラフト。

これらのプロトコルの使用に関する詳細なガイダンスについては、RTMP および WHIP のドキュメントを参照してください。

サポートされているメディア仕様

- オーディオ入力形式
  - コーデック: AAC-LC for RTMP および Opus for WHIP
  - チャネル: 2 (ステレオ) または 1 (モノ)
  - サンプルレート: 44.1 kHz または 48 kHz
  - 最大ビットレート: 160 Kbps
- ビデオ入力形式
  - コーデック: H.264
  - H.264 プロファイル: ベースライン
  - IDR 間隔: 1 秒または 2 秒
  - フレームレート: 10 ~ 60 FPS
  - B フレーム: 0

    注: IVS Broadcast SDK では、デフォルトで B フレームが有効になっていますが、バージョン 1.25.0 以降では、IVS ステージにブロードキャストするときに B フレームが自動的に無効になります。他の RTMP エンコーダーを使用してリアルタイムストリーミングを行うには、開発者は B フレームを無効にする必要があります。他の RTMP エンコーダーを使用する開発者が B フレームを無効にしなかった場合、そのストリームは切断されます。
  - 最大解像度: 720p 最小解像度: 160p
  - 最大ビットレート: 8.5 Mbps

    注: シングルトラック RTMP ストリームの場合、この制限はそのトラックに適用されます。拡張 RTMP を使用して公開されたマルチトラックビデオの場合、制限はすべてのビデオトラックの合計ビットレートに適用されます。
  - エンコーダー設定: H.264 エンコーダーには `veryfast` と `zerolatency` 設定を使用することをお勧めします。また、`sliced_threads` x264 オプションは `zerolatency` プリセットに含まれているため、無効にすることをお勧めします。例えば、FFmpeg を使用する場合、コマンドには以下を含める必要があります。`-preset:v veryfast -tune zerolatency -x264-params sliced-threads=0`

IVS RTMP 配信 | リアルタイムストリーミング

このドキュメントでは、RTMP を使用して IVS ステージに配信するプロセスの概要を説明します。さまざまな取り込みオプションの詳細については、「ストリームの取り込み」ドキュメントを参照してください。

前提条件

ステージの作成

ステージを作成するには、以下のコマンドを使用します。

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

レスポンスを含む詳細については、「CreateStage」を参照してください。

重要： レスポンスで、RTMP エンドポイントと RTMPS エンドポイントの両方を一覧表示する `endpoints` フィールドを書き留めます。これらは RTMP エンコーダーのセットアップに必要です。

取り込み設定を作成する

RTMPS を使用してステージに配信するには、まず取り込み設定を作成し、ステージに関連付ける必要があります。ステージに配信すると (取り込み設定のストリームキーとステージの RTMP エンドポイントを使用)、メディアは参加者としてステージに配信されます。`userId` およびカスタム `attributes` を指定するオプションがあり、ステージに接続する参加者に関連付けられます。

```
aws ivs-realtime create-ingest-configuration \
  --name 'test' \
  --stage-arn arn:aws:ivs:us-east-1:123456789012:stage/8faHz1SQp0ik \
  --user-id '123' \
  --ingest-protocol 'RTMPS'
```

レスポンスを含む詳細については、「CreateIngestConfiguration」を参照してください。

取り込み設定を作成するときは、事前に特定のステージ ARN に関連付けることができます。この関連付けがない場合、ストリームキーは使用できません。また、インジェスト設定 (`stageArn` フィールドを含む) は UpdateIngestConfiguration オペレーションを介して更新できるため、異なるステージで同じ設定を再利用できます。

注: 取り込み設定 `insecureIngest` フィールドはデフォルトで `false` になり、RTMPS を使用する必要があります。RTMP 接続は拒否されます。RTMP を使用する必要がある場合は、`insecureIngest` を `true` に設定します。RTMP を必須とする特定の検証済みのユースケースがない限り、RTMPS の使用をお勧めします。

RTMP シングルトラックビデオ

ここでは、OBS Studio の使用方法を示しますが、IVS メディア仕様を満たす任意の RTMP エンコーダを使用できます。

OBS ガイド

1. ソフトウェアをダウンロードしてインストールします: https://obsproject.com/download。
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 [カスタム] を選択します。
3. [サーバー] には、ステージから RTMP または RTMPS エンドポイントを入力します。
4. ストリームキー には、取り込み設定から `streamKey` を入力します。
5. ビデオ設定を通常どおりに設定します。いくつかの制限があります。
   a. IVS Real-Time Streaming 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。CPU 使用率プリセットを [veryfast] に、レイテンシーを最小にするために調整を [zerolatency] に設定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、ビットレートを 2.5 Mbps 未満に維持することをお勧めします。これにより、低帯域幅接続の視聴者が視聴できるようになります。
   d. B フレームを含むストリームは自動的に切断されるため、B フレームを無効にします。次のいずれかを行います。
      - x264 オプションで、`bframes=0 sliced-threads=0` と入力します。
      - オプション (NVENC の場合など) の場合は、B フレームを 0 に設定します。

   注意: RTMP ストリームにはオーディオトラックとビデオトラックの両方が含まれている必要があります。含まれていない場合、ストリームは切断されます。
6. 次に [ストリーミングを開始] を選択します。

重要: エンコーダーの最大ビットレートが 8.5 Mbps に設定されている場合、パブリッシャーはセッションから消えることがあります。これは、最大ビットレート設定が目標のみであり、エンコーダーが目標を超えることがあるためです。これを防ぐには、エンコーダーの最大ビットレートを 6 Mbps などに設定します。

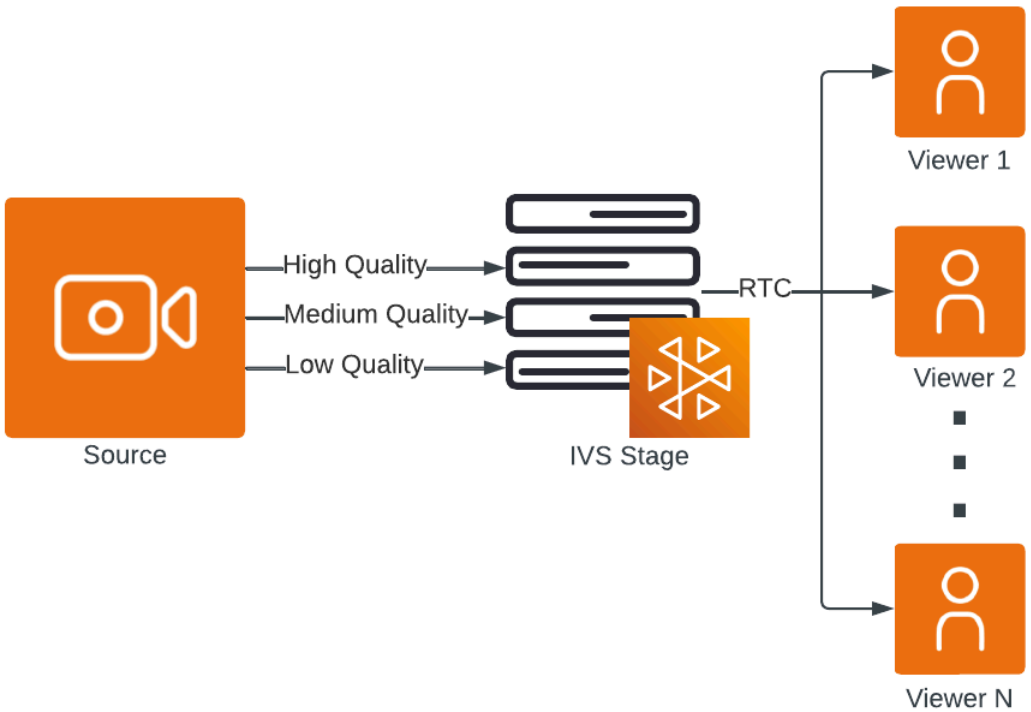
E-RTMP マルチトラックビデオ

IVS は、E-RTMP (拡張リアルタイムメッセージングプロトコル) のマルチトラックビデオ機能をサポートするので、単一の RTMP ストリームで複数のビデオ品質を IVS ステージに配信できます。これによりアダプティブビットレートストリーミングが可能になり、サブスクライバーはネットワーク接続に最適な品質で自動的に視聴できます。

取り込まれると、さまざまなビデオ品質がサイマルキャストレイヤーとしてサブスクライバーに配信されます。サブスクライバーが受信するレイヤーを設定するには、リアルタイムストリーミング SDK ガイド (Android、iOS、Web) のサイマルキャストによるレイヤードエンコーディングに関するセクションを参照してください。

サンプルコードについては、GitHub の aws-samples/sample-amazon-ivs-multitrack-video を参照してください。

この図は、マルチトラックビデオを使用した配信の仕組みを示しています。



OBS ガイド

1. OBS Studio をダウンロードしてインストールします。
   a. Windows: マルチトラックビデオは OBS Studio 30.2 以降でサポートされています。
   b. macOS: マルチトラックビデオは、OBS Studio 31.1 Beta (Apple Silicon のみ) 以降でサポートされています。
   c. ダウンロード: https://obsproject.com/download。
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 [Amazon IVS] を選択します。
3. [サーバー] は [自動] のままにします。
4. ストリームキー には、取り込み設定から `streamKey` を入力します。
5. [Multitrack Video] セクションで、[Enable Multitrack Video] を選択します。
6. [ビデオ] パネルで、目的の [Base (Canvas Resolution)] と [Output (Scaled) Resolution] を設定します。IVS Real-Time Streaming は、最大 720p の入力をサポートします。これらの制限を超えると、ストリームは切断されます。

   マルチトラックビデオが有効な場合、ビデオトラックの数、ビットレート、キーフレーム間隔などの設定は、デバイスの容量に基づいて自動的に設定されます。
7. [ストリーミングを開始] を選択します。

FFmpeg による配信

FFmpeg を使用して、RTMP 経由での IVS リアルタイムストリーミングにライブビデオと音声を配信できます。無料のオープンソースプロジェクトである FFmpeg は、ビデオ、音声、およびその他のマルチメディアコンテンツを処理するための包括的な一連のソフトウェアライブラリで構成されています。

以下のコマンド例は、色のパターンとトーンが含まれるストリームを配信します。

```
ffmpeg \
 -re \
 -f lavfi -i testsrc=d=300:s=1280x720:r=60,format=yuv420p \
 -f lavfi -i sine=f=440:b=4:d=300 \
 -c:v libx264 \
 -b:v 2500k \
 -g 60 -bf 0 \
```

```
-profile:v baseline \
-preset veryfast \
-tune zerolatency \
-x264opts sliced-threads=0 \
-c:a aac \
-ac 2 \
-b:a 160k \
-ar 48000 \
-f flv \
rtmps://$INGEST_ENDPOINT/app/$STREAM_KEY
```

この例では、$INGEST_ENDPOINT と $STREAM_KEY を IVS コンソールまたは API の独自の値に置き換えてください。

この設定は、H.264 ビデオ (ベースラインプロファイル、B フレームなし、スライスされたスレッドなし) や AAC 音声などの IVS リアルタイムストリーミングでサポートされているメディア仕様に適合しています。

ステージへのプライベート取り込み

インターフェイス VPC エンドポイントを使用して、Amazon VPC 内のリソースまたは Direct Connect からステージに RTMP(S) ストリームと E-RTMP(S) ストリームを配信できます。これによって VPC と IVS 間のプライベート接続が可能になり、AWS ネットワーク内に取り込みトラフィックを保持できます。IVS のインターフェイス VPC エンドポイントを設定および構成するには、「IVS 低レイテンシーストリーミングユーザーガイド」の「IVS Private Ingest」を参照してください。

IVS WHIP パブリッシュ | 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 Real-Time Streaming に公開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 (IVS Broadcast SDK との) 代替を提供します。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、開発者は、IVS Web Broadcast SDK を使用してブラウザを直接公開したり、ストリーマーがデスクトップで OBS を使用してより強力なツールを使用したりするなど、汎用性の高いオプションを利用できます。

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 は使用でき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。

WHIP の要件:

- SDP オファーには、オーディオのみを公開している場合でも H.264 ビデオトラックが含まれている必要があります。オファーにビデオトラックが含まれていない場合、接続は拒否されます。
- グローバル WHIP エンドポイント (https://global.whip.live-video.net) は 307 一時リダイレクトを返します。WHIP クライアントは WHIP 仕様の要求に従って、307 リダイレクトを正しく処理し、リダイレクトされたリクエストにヘッダーを保持する必要があります。

OBS ガイド

OBS はバージョン 30 時点で 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. 公開機能を使用して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、参加者トークンの有効期限は 12 時間ですが、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. [Bearer Token] には、ステップ 1 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。いくつかの制限があります。
   a. IVS Real-Time Streaming 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。CPU 使用率プリセットを [veryfast] に、レイテンシーを最小にするために調整を [zerolatency] に設定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、ビットレートを 2.5 Mbps 未満に維持することをお勧めします。これにより、低帯域幅接続の視聴者が視聴できるようになります。

6. [ストリーミングを開始] を押します。

注: OBS の WHIP で発生する可能性のある品質上の問題 (断続的なビデオフリーズなど) を認識しています。これらは通常、ブロードキャスターのネットワークが不安定な場合に発生します。本番ライブストリームに使用する前に、OBS で WHIP をテストすることをお勧めします。ブロードキャストビットレートを下げると、これらの問題の発生を減らすこともできます。

IVS 参加者のレプリケーション | リアルタイムストリーミング

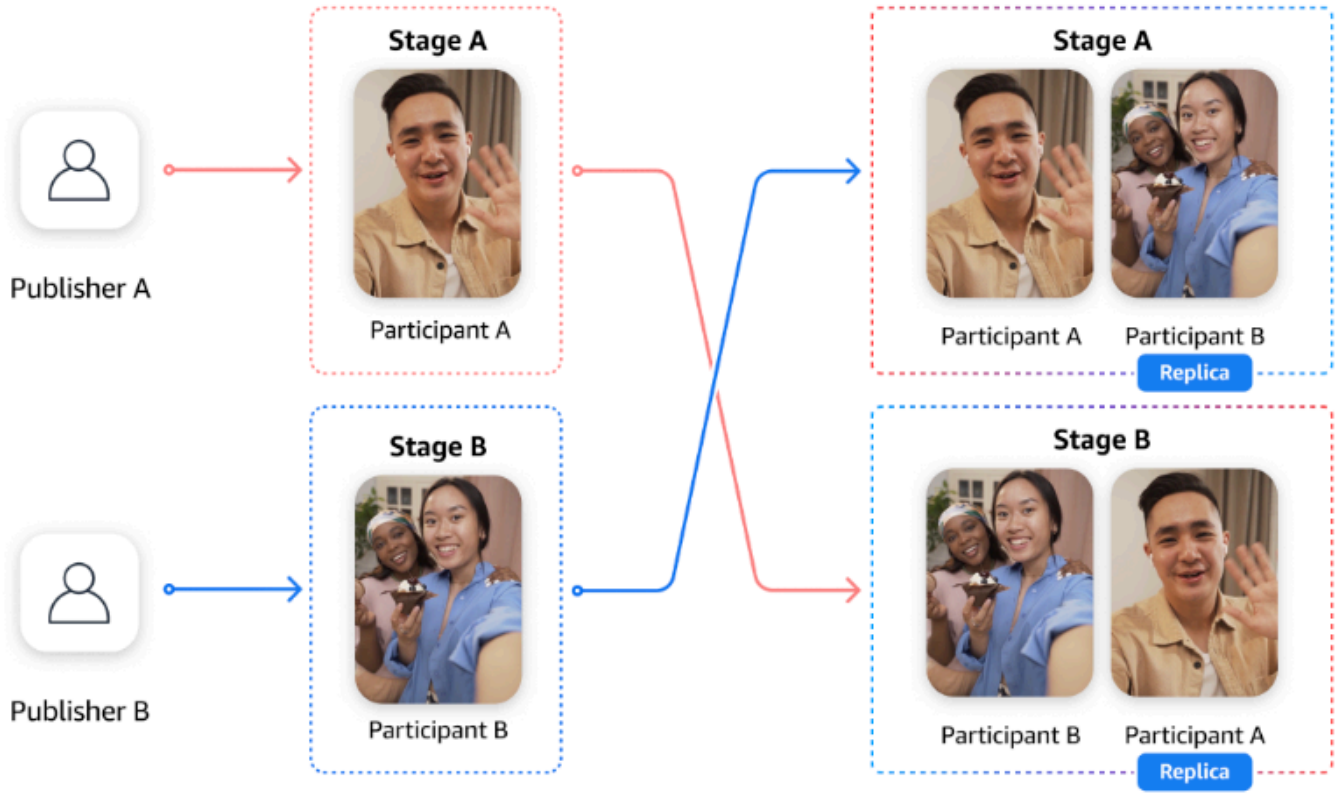
参加者のレプリケーションでは、あるステージから別のステージに参加者をコピーできます。これは、同じ参加者を複数のステージに同時に表示し、ステージ間のインタラクションを有効にする場合に便利です。

ソーシャルライブストリーミングアプリケーションの一般的なユースケースは、VS モードと呼ばれる対戦形式です。この形式では、2 人の配信者が一時的にマッチングされ、リアルタイムで相互にやり取りできるようになります。また、各配信者の視聴者は両方の配信者を同時に視聴することができます。

主要なコンセプト:

- ソースステージ — 参加者が最初に参加したステージ。レプリケーションのソースとして使用されます。
- 宛先ステージ — 参加者がレプリケートされるステージ。
- レプリケートされる参加者 — 1 つ以上の宛先ステージにレプリケートされるステージの参加者。
- レプリカ参加者 — 別のステージ (ソースステージ) からレプリケートされる宛先ステージの参加者。

参加者のレプリケーションの使用



前提条件

参加者のレプリケーションを使用するには、少なくとも 2 つのステージがすでに作成されている必要があります。たとえば、上記のシナリオでは、2 人のアクティブなパブリッシャーが存在しています。

1. ステージ A に接続された参加者 A
2. ステージ B に接続された参加者 B

対戦形式をサポートするために、一時的に参加者 A をステージ B に、参加者 B をステージ A にレプリケートします。

参加者のレプリケーションを開始する

参加者をレプリケートするには、StartParticipantReplication オペレーションを使用します。これはレプリケーション方向ごとに 1 回呼び出す必要があります。

参加者 A をステージ B にレプリケートします。

```
aws ivs-realtime start-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --participant-id participant-a-id \
  --reconnect-window-seconds 10
```

参加者 B をステージ A にレプリケートします。

```
aws ivs-realtime start-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --participant-id participant-b-id \
  --reconnect-window-seconds 10
```

レプリケーションが開始されると、StopParticipantReplication オペレーションを使用して明示的に停止するまで、参加者はレプリケートされたままになります。レプリケートされた参加者は、切断された後に `reconnectWindowSeconds` で指定された時間内に再接続した場合、自動的にソースステージと宛先ステージの両方に再び表示されます。`reconnectWindowSeconds` のデフォルト値は 0 です。

参加者のレプリケーションを停止する

レプリケーションを停止するには、StopParticipantReplication オペレーションを呼び出します。

ステージ A からステージ B への参加者 A のレプリケーションを停止します。

```
aws ivs-realtime stop-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --participant-id participant-a-id
```

ステージ B からステージ A への参加者 B のレプリケーションを停止します。

```
aws ivs-realtime stop-participant-replication \
  --source-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageB \
  --destination-stage-arn arn:aws:ivs:us-east-1:123456789012:stage/StageA \
  --participant-id participant-b-id
```

IVS 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、特定の AWS リージョンでアカウント単位で適用されます。

Service Quotas 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、Service Quotas 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

オペレーションのタイプ	運用	デフォルト
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
IngestConfiguration	`CreateIngestConfiguration`	5 TPS
IngestConfiguration	`DeleteIngestConfiguration`	5 TPS
IngestConfiguration	`GetIngestConfiguration`	5 TPS
IngestConfiguration	`ListIngestConfigurations`	5 TPS

オペレーションのタイプ	運用	デフォルト
IngestConfiguration	UpdateIngestConfiguration	5 TPS
MediaEncoder	CreateEncoderConfiguration	5 TPS
MediaEncoder	DeleteEncoderConfiguration	5 TPS
MediaEncoder	GetEncoderConfiguration	5 TPS
MediaEncoder	ListEncoderConfigurations	5 TPS
ParticipantReplication	ListParticipantReplicas	5 TPS
ParticipantReplication	StartParticipantReplication	5 TPS
ParticipantReplication	StopParticipantReplication	5 TPS
パブリックキー	DeletePublicKey	3 TPS
パブリックキー	GetPublicKey	3 TPS
パブリックキー	ImportPublicKey	3 TPS
パブリックキー	ListPublicKeys	3 TPS
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

オペレーションのタイプ	運用	デフォルト
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	引き上げ可能	説明
コンポジションの送信先	2	不可	Composition 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	不可	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	20	あり	アカウントあたりの同時 Composition リソースの最大数。

リソースまたは機能	デフォルト	引き上げ可能	説明
ステージあたりのコンポジション数	5	あり	ステージあたりの同時 Composition リソースの最大数。
同時参加者レプリケーション	5	不可	AWSリージョンのすべてのステージにおける 1 参加者あたりの同時レプリケーションの最大数。
同時パブリッシャー	1,000	あり	AWS リージョンのすべてのステージにわたって公開できる参加者の最大数。
同時サブスクリプション	20,000	あり	AWS リージョンのすべてのステージにわたるパブリッシャーからサブスクライバーへの同時接続の最大数。
EncoderConfigurations	20	あり	アカウントあたりの EncoderConfiguration リソースの最大数。
IngestConfigurations	100	あり	アカウントあたりの IngestConfiguration リソースの最大数。
参加者のダウンロードビットレート	8.5 Mbps	不可	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。
参加者がビットレートを公開する	8.5 Mbps	不可	ステージにストリーミングできる、1 秒あたりの最大ビット数です。

リソースまたは機能	デフォルト	引き上げ可能	説明
参加者の公開期間またはサブスクライブ期間	24	不可	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	不可	参加者が公開する動画の最大解像度。
PublicKeys	3	不可	AWS リージョンあたりのパブリックキーの最大数。
ステージ参加者 (パブリッシャー)	12	不可	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	あり	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	1,000	あり	AWS リージョンあたりのステージの最大数。
StorageConfigurations	5	あり	アカウントあたりの StorageConfiguration リソースの最大数。

IVS Real-Time Streaming の最適化

ユーザーが IVS Real-Time Streaming 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。

IVS SDK を使用すると、ストリームの最大ビットレート、フレームレート、解像度を設定できます。パブリッシャーにネットワーク輻輳が発生すると、SDK は自動的に調整してビットレート、フレームレート、解像度を下げることによってビデオ品質を低下させます。Android および iOS では、輻輳が発生した際にデグレーション設定を選択できます。サイマルキャストでレイヤードエンコーディングを有効にしても、デフォルト設定のままにしても、同じ動作が適用されます。

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

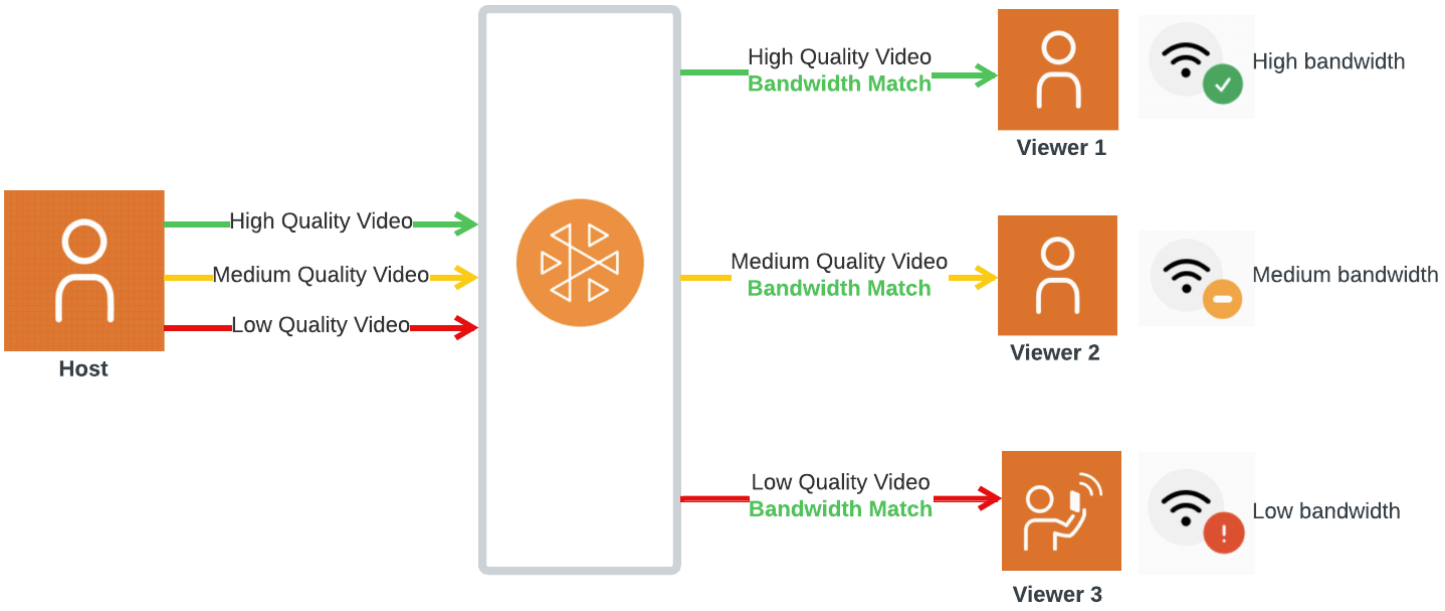
この機能は、以下のクライアントバージョンでのみサポートされています。

- iOS とAndroid 1.18.0 以降
- Web 1.12.0 以降

IVS リアルタイム Broadcast SDK を使用する場合、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ネットワークに最適な品質に自動的に適応または変更します。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 と Edge 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	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
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
低レイヤー: • 最大ビットレート: 100,000 bps	低レイヤー: • 最大ビットレート: 200,000 bps

モバイル (Android、iOS)	Web (Chrome)
• フレームレート: 15 fps	• フレームレート: 15 fps

レイヤーの解像度

中レイヤーと低レイヤー解像度は、同じアスペクト比を維持するために、高レイヤーから自動的にスケールダウンされます。

解像度が上記のレイヤーに近づくと、中レイヤーと低レイヤーは除外されます。例えば、設定された解像度が 320x180 の場合、SDK は低解像度レイヤーを送信しません。

次の表は、さまざまな設定済み解像度に対して生成されたレイヤーの解像度を示しています。リストされている値は横向きですが、縦向きのコンテンツには縦横逆に適用されます。

入力解像度	出力レイヤーの解像度: モバイル	出力レイヤーの解像度: Web
720p (1280x720)	高 (1280x720) 低 (320x180)	高 (1280x720) 中 (640x360) 低 (320x180)
540p (960x540)	高 (960x540) 低 (320x180)	高 (960x540) 低 (320x180)
360p (640x360)	高 (640x360) 低 (360x180)	高 (640x360) 低 (360x180)
270p (480x270)	高 (480x270)	高 (480x270)
180p (320x180)	高 (320x180)	高 (320x180)

上記にマッピングされていないカスタム入力解像度については、次のツールを使用して計算できます。

サイマルキャストによるレイヤードエンコーディングの設定 (パブリッシャー)

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、クライアントでこの機能を有効にしている必要があります。有効にすると、パブリッシャーによるアップロード帯域幅の使用が増加し、視聴者側の動画フリーズを減らせる可能性があります。

Android

```
// Enable Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
// Enable Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

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

// Other Stage implementation code
```

Web

```
// Enable Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})

// Other Stage implementation code
```

個々のレイヤーの設定の詳細については、各 Broadcast SDK ガイド: Android、iOS、および Web の「レイヤードエンコーディングの設定 (パブリッシャー) 」を参照してください。

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 (サブスクライバー)

サブスクライバーが受信するレイヤーの設定については、リアルタイムストリーミング SDK ガイドの「サイマルキャストによるレイヤードエンコーディング」セクションを参照してください。

- Android Broadcast SDK
- IVS Broadcast SDK
- Web Broadcast SDK

サブスクライバー設定を使用すると、InitialLayerPreference を定義できます。これにより、最初に配信されるビデオの品質と preferredLayerForStream が決定されます。これにより、ビデオの再生中に選択されるレイヤーを決定します。レイヤーの変更、適応の変更、レイヤーの選択が行われたときに通知するイベントやストリームメソッドがあります。

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

```
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
```

サブスクライバージッターバッファ MinDelay の変更

サブスクライブしている参加者のジッターバッファの最小遅延を変更するには、カスタム `subscribeConfiguration` を使用できます。ジッターバッファは、再生が開始される前に保存されるパケットの数を決定します。最小遅延は、保存する必要があるデータの最小量の目標を表します。最小遅延を変更すると、パケット損失/接続の問題に直面したときに再生の回復力を高めることができます。

ジッターバッファのサイズを大きくすると、再生が開始されるまでの遅延も大きくなります。最小遅延を増やすと、回復力が向上しますが、代償としてビデオになるまでの時間に影響が出ます。再生中の最小遅延を長くすると、同様の効果があります。ジッターバッファがいっぱいになるまでの間、再生は短時間一時停止します。

耐障害性を高める必要がある場合は、再生を開始する前に、最小遅延プリセット MEDIUM から開始し、サブスクライブ設定を設定することをお勧めします。

最小遅延は、参加者がサブスクライブ専用である場合にのみ適用されることに注意してください。参加者が自分で公開している場合、最小遅延は適用されません。これは、複数のパブリッシャーが追加の遅延なく相互に話せるようにするために行われます。

以下の例では、最小遅延プリセット MEDIUM を使用しています。使用可能なすべての値については、SDK リファレンスのドキュメントを参照してください。

Web

```
const strategy = {
   subscribeConfiguration: (participant) => {
      return {
         jitterBuffer: {
            minDelay: JitterBufferMinDelay.MEDIUM
         }
      }

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

Android

```
@Override
public SubscribeConfiguration subscribeConfigrationForParticipant(@NonNull Stage stage,
 @NonNull ParticipantInfo participantInfo) {
    SubscribeConfiguration config = new SubscribeConfiguration();


 config.jitterBuffer.setMinDelay(JitterBufferConfiguration.JitterBufferDelay.MEDIUM());

    return config;
}
```

iOS

```
func stage(_ stage: IVSStage, subscribeConfigurationForParticipant participant:
 IVSParticipantInfo) -> IVSSubscribeConfiguration {
    let config = IVSSubscribeConfiguration()
```

```
    try! config.jitterBuffer.setMinDelay(.medium())

    return config
}
```

推奨される最適化

シナリオ	推奨事項
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)。
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。

ネットワーク要件 | Real-Time Streaming

Amazon IVS Real-Time Streaming は、メディアとデータの送信に WebRTC プロトコルと WebSocket プロトコルを活用しています。シームレスなエクスペリエンスを確保するには、以下にリストされているデスティネーションとポートにアクセスできる必要があります。これらのデスティネーションに対するインバウンドまたはアウトバウンドのトラフィックに制限があると、IVS Real-Time Streaming の機能が妨げられる可能性があります。

このネットワークテストツールを使用して、ネットワークが正しく設定されており、必要なデスティネーションとポート間を行き来するトラフィックがブロックされていないことを確認できます。

共通

宛先	ポート
*.live-video.net	TCP: 443

メディア

IVS Real-time Streaming はデフォルトで、メディアの送信に UDP を活用し、低レイテンシーで高性能なストリーミングを確保しています。メディアにサブスクライブしている参加者に対して UDP がブロックされている場合、TCP はフォールバックとして使用されます。TCP フォールバックは公開ではサポートされていないため、UDP トラフィックが完全にブロックされると、公開は失敗します。

宛先	ポート
ip-ranges.json の `IVS_REALTIME` サービスにリストアップされているすべてのサブネットは、それらの `region` や、ユーザーが選択した AWS リージョンに関係なく、アクセス可能である必要があります。参加者は、いずれかのサブネットに自動的に接続される場合があります。詳細については、「グローバルソリュー	UDP:3478 UDP:443 TCP:3478 (フォールバック) TCP: 443

宛先	ポート
ション、リージョナルコントロール」を参照してください。	

IVS のコスト |リアルタイムストリーミング

IVS のコストの詳細については、「IVS 料金表」を参照してください。

- ステージへのサブスクライブと公開 - サブスクライブと公開はリソースを消費するため、ステージに接続した時間単位の料金が発生します。
- 録画 - 個々の参加者の録画には Amazon IVS の追加料金はかかりませんが、合成録画には、エンコードされたビデオの時間単位の料金が発生します。どちらの録画オプションにも、標準の S3 ストレージとリクエストコストが発生します。サムネイルでは追加の IVS 料金は発生しません。
- 参加者のレプリケーション — レプリカの参加者には、通常の参加者と同じ料金が請求されます。

  たとえば、ステージ A に参加者 A、ステージ B に参加者 B がいる場合、2 人の参加者分の料金が発生します。

  参加者 A がステージ B にレプリケートされると、接続されている参加者は 3 人 (参加者 A、参加者 B、および参加者 A のレプリカ) になります。レプリケーション中は、3 人分の参加者料金が発生します。

詳細については、「IVS 料金表」を参照してください。

IVS リソースおよびサポート | リアルタイムストリーミング

このドキュメントでは、Amazon IVS Real-Time Streaming の使用をサポートするリソースの一覧を示します。

デモおよびその他のリソース

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 IVS を体験するための専用サイトです。デモとコードサンプルについては、https://ivs.rocks/examples を参照してください。

DEV Community サイトの Amazon IVS ページには、豊富なデモやブログ記事があります。たとえば、「Getting Started with Amazon Interactive Video Service」は、IVS の使用に関する初心者向けの一連の記事です。記事では、投稿に埋め込まれたインタラクティブなデモを使用して、IVS API のステップバイステップのウォークスルーを紹介しています。すべてのデモは、埋め込まれた CodePen を介して投稿から直接実行できます。

AWS ブログサイトには、さまざまなトピックに関する IVS ブログ投稿が多数あります。ページの右側にある [製品またはソリューション] > [メディアサービス] > [Amazon インタラクティブビデオサービス] を選択して、IVS を絞り込みます。

GitHub の iOS および Android 用の IVS Real-Time Streaming デモでは、開発者向けに、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。ゲストスポットまたはバーサス (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構築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AWS - 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 Discussion Forums またはサポート用のリンクをご利用ください。

IVS の用語集

「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	Chat
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、「サービス認可リファレンス」を参照してください。	✓	✓	✓

言葉	説明	LL	RT	Chat
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	Chat
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	Chat
[チャネル]	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。	✓	✓	
CloudFront	Amazon が提供する CDN サービス。	✓		

言葉	説明	LL	RT	Chat
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 CloudTrail を使用した IVS.API コールのログ作成」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」と「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。	✓	✓	
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。	✓	✓	✓

言葉	説明	LL	RT	Chat
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。	✓		
カスタムオーディオソース	IVS Broadcast SDK が提供するインターフェイス。デバイスの内蔵マイクに限定されず、アプリケーションが独自の音声入力を行えるようにします。		✓	
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。	✓	✓	
カスタム参加者の順序付け	参加者トークンのカスタム属性値に基づいて、グリッドレイアウトと PiP レイアウトの両方でステージ参加者の配置を有効にします。		✓	
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。	✓	✓	
E-RTMP	拡張 RTMP プロトコル。IVS は、マルチトラックビデオに必要な E-RTMP の機能をサポートしています。	✓		

言葉	説明	LL	RT	Chat
イベント	IVS が AmazonEventBridge モニタリングサービスに発行する自動通知。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。「IVS 低レイテンシーストリーミングで Amazon EventBridge を使用する」および「IVS リアルタイムストリーミングで Amazon EventBridge を使用する」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	Chat
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	ステージを作成するリアルタイムのユーザー。		✓	
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	Chat
JSON	JavaScript オブジェクト表記とは、人が読み取れるテキストを使用して、属性と値のペアと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、オープンなスタンダードファイル形式。	✓	✓	✓
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、グラストゥグラスのレイテンシー	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 グラストゥグラスレイテンシーとは、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	Chat
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。	✓		
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、クライアントサイドまたはサーバーサイド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。		✓	

言葉	説明	LL	RT	Chat
マルチトラックビデオ	ブロードキャスターソフトウェアツールが GPU 搭載のコンピュータから複数のビデオ品質を直接エンコードおよびストリーミングできるようにします。「Amazon IVS のマルチトラックビデオ」を参照してください。	✓		
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	オリジンアクセスコントロールは S3 バケットへのアクセスを制限するメカニズムで、記録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します。	✓		
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	パブリッシャーまたはサブスクライバーとしてステージに接続されたリアルタイムのユーザー。		✓	
参加順序	ステージ参加者がグリッドレイアウトと PiP レイアウトに配置される順序。		✓	
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	Chat
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可メカニズム。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、再生用の視聴者認可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キーとプライベートキーのペアです。「IVS プライベートチャネルの設定」の「IVS 再生キーの作成またはインポート」、ならびに「IVS 低レイテンシー API リファレンス」の「再生キーペアのオペレーション」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 に基づく認可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。「IVS プライベートチャネルの設定」の「IVS 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。	✓		
プライベート取り込み	AWS PrivateLink を搭載したインターフェイス VPC エンドポイントを使用して、Amazon VPC と IVS 間の安全なプライベート接続を有効にします。「IVS プライベート取り込み」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	Chat
パブリッシャー	ビデオやオーディオをステージに公開するリアルタイムイベントの参加者。「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。		✓	
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 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。	✓	✓	✓
Regions	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。	✓	✓	✓
解決策	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。	✓	✓	

言葉	説明	LL	RT	Chat
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。	✓	✓	✓
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。	✓		

言葉	説明	LL	RT	Chat
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。 IVS Player SDK は IVS ストリームの再生用です。IVS アーキテクチャを活用し、IVS 低レイテンシー再生用に最適化されています。ウェブ、Android、iOS 用の IVS プレーヤー SDK があります。 IVS Broadcast SDK 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ものです。この SDK は IVS アーキテクチャを活用し、IVS とともに継続的に改善されています。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。低レイテンシーおよびリアルタイムストリーミング用のウェブ、Android、iOS 用の IVS Broadcast SDK があります。	✓	✓	✓
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。		✓	

言葉	説明	LL	RT	Chat
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。	✓	✓	✓
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。	✓	✓	✓
サービスにリンクされたロール	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。	✓		

言葉	説明	LL	RT	Chat
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の開始」の「ステージを作成する」を参照してください。		✓	
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可に使用されます。 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。	✓		

言葉	説明	LL	RT	Chat
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。	✓	✓	
サブスクライバー	ステージパブリッシャーのビデオやオーディオを受信するリアルタイムイベントの参加者。「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。	✓	✓	✓

言葉	説明	LL	RT	Chat
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。	✓		
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。	✓		
トークン交換	IVS Broadcast SDK が提供するインターフェイス。参加者が再接続することなく、参加者トークン機能をアップグレードまたはダウングレードし、トークン属性を更新できます。これにより、参加者がサブスクライブ専用機能から開始し、後でパブリッシュ機能が必要になる共同ホスティングなどのシナリオが可能になります。		✓	

言葉	説明	LL	RT	Chat
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。	✓	✓	
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	Chat
表示	アクティブにメディアファイル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。直接ピアツーピア通信が可能になるため、ウェブページ内でオーディオや動画の通信が可能になり、プラグインをインストールしたりネイティブアプリをダウンロードしたりする必要がなくなります。 WebRTC の背後にある技術はオープンなウェブ規格として実装されており、すべての主要なブラウザで通常の JavaScript API として、または Android や iOS などのネイティブクライアント用のライブラリとして利用できます。	✓	✓	

言葉	説明	LL	RT	Chat
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。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 (IVS Broadcast SDK との) 代替を提供します。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。 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 は使用でき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。 「IVS WHIP サポート (リアルタイムストリーミング)」を参照してください。		✓	
WSS	WebSocket Secure は、暗号化された TLS 接続を介して WebSockets を確立するためのプロトコルです。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。			✓

IVS ドキュメント履歴 | リアルタイムストリーミング

次の表に Amazon IVS Real-Time Streaming ドキュメントの重要な変更を示します。新しいリリースをお知らせするとともに、お客様からお寄せいただいたフィードバックに対応するためにドキュメントは頻繁に更新されています。

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
Broadcast SDK: Web 1.33.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2026 年 3 月 12 日
Broadcast SDK: Android 1.40.0、iOS 1.40.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2026 年 3 月 12 日
EventBridge	4 つのイベント (送信先の失敗、セッションの失敗、記録開始の失敗、記録終了の失敗) について説明を更新し、`error_code` フィールドおよびエラーコードと理由の表を追加しました。	2026 年 2 月 27 日
Broadcast SDK: Android 1.39.0、iOS 1.39.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、	2026 年 2 月 13 日

	バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 iOS 統合では、CocoaPods は使用されなくなりました。関連するドキュメントの変更が、次の場所で行われました。 • IVS リアルタイムストリーミングの開始 –「ステップ 4: IVS Broadcast SDK を統合する」 >「iOS」 • 「iOS Broadcast SDK ガイド」 –「ライブラリのインストール」	
Broadcast SDK: Web 1.32.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2026 年 2 月 12 日
Service Quotas	3 つの ParticipantReplication オペレーション (List/Start/Stop) の TPS 値を追加しました。	2026 年 1 月 30 日
Broadcast SDK: Android 1.38.0、iOS 1.38.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2026 年 1 月 13 日

Broadcast SDK: Android 1.37.1	リアルタイムストリーミングのブロードキャスト SDK ガイドのバージョン番号およびアーティファクトのリンクが更新されました (Android)。「リリースノート」を参照してください。	2025 年 12 月 11 日
ブロードキャスト SDK – カスタムオーディオソース	この新しい Chat ページを追加しました。	2025 年 12 月 10 日
参加者トークン交換	IVS Broadcast SDK の下に新しいトークン交換ページを追加しました。 「IVS Real-Time Streaming で Amazon EventBridge を使用する」で、IVS Stage Update イベント「トークン交換」を追加しました。また、「例: Stage Update」にトークン交換の例を追加しました。 API の変更については、API リファレンス表を参照してください。	2025 年 12 月 9 日
Broadcast SDK: Web 1.31.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 12 月 5 日

Broadcast SDK: Android 1.37.0、iOS 1.37.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 12 月 5 日
CloudWatch メトリクス	リアルタイムストリーミングのモニタリング > CloudWatch メトリクスにおいて、新たに多数の DroppedFrames、PublishBitrate、SubscribeBitrate メトリクスが追加されました。また、いくつかのメトリクスの説明を更新しました。	2025 年 11 月 18 日
IPR の同期	「個々の参加者の録画」に、「複数の参加者の録画を同期させる」を追加しました。	2025 年 11 月 7 日
サーバーサイドコンポジション	「既知の問題と回避策」を追加しました。	2025 年 10 月 30 日
Broadcast SDK: Web 1.30.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 10 月 30 日

Broadcast SDK: Android 1.36.0、iOS 1.36.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 また、「メッセージを埋め込む」に「Android」と「iOS」という新しいセクションを追加しました。	2025 年 10 月 30 日
サーバーサイドコンポジション	「Composition のライフサイクル」を更新しました (コンポジションが自動シャットダウンを実行する場合)。	2025 年 10 月 27 日
RTMP	「FFmpeg による配信」を追加しました。	2025 年 10 月 27 日
コンポジションのクォータを更新	Service Quotas の「その他のクォータ」で、「アカウントあたりの同時 Composition リソースの最大数」のクォータを 5 から 20 に更新しました。	2025 年 10 月 14 日
Web Broadcast SDK の更新	「IVS Web Broadcast SDK での配信とサブスクライブ」>「WebRTC 統計を取得する」セクションを書き換えました。	2025 年 10 月 3 日

CloudWatch メトリクス	`ConcurrentPublishers` および `ConcurrentSubscriptions` メトリクスについて説明を更新し、ディメンションを削除しました。	2025 年 10 月 2 日
Broadcast SDK: Web 1.29.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 10 月 2 日
Broadcast SDK: Android 1.35.0、iOS 1.35.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 10 月 2 日
CloudWatch メトリクス	DownloadPacketLoss メトリクスをいくつか追加しました。	2025 年 9 月 23 日
SSC カスタム参加者の順序付け	`participantOrderAttribute` および「カスタム参加者の順序付け」の追加など、「サーバーサイドコンポジション」にさまざまな変更を加えました。 API の変更については、API リファレンス表を参照してください。	2025 年 9 月 16 日

Broadcast SDK: Android 1.34.0、iOS 1.34.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 9 月 11 日
ステージへのプライベート取り込み	インターフェイス VPC エンドポイントのリリースの新しいセクション。	2025 年 9 月 10 日
Broadcast SDK: Web 1.28.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 9 月 4 日
Broadcast SDK: iOS	iOS Broadcast SDK (1.33.0) の最新のリリースノートに 1.32.1 iOS Broadcast SDK の修正を追加しました。	2025 年 8 月 21 日
Broadcast SDK: ウェブサイト	「開始方法」の更新 > 「スクリプトタグの使用」および「npm の使用」の両方の「インポート」。	2025 年 8 月 8 日
Broadcast SDK: Web 1.27.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 8 月 7 日

Broadcast SDK: Android 1.33.0、iOS 1.33.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 iOS Broadcast Guide に「Recommended: Integrate the Player SDK (Swift Package Manager)」が追加され、CocoaPods 統合に関する既存の情報が更新されました。	2025 年 8 月 7 日
Broadcast SDK: 混合デバイス	これは新しいドキュメントです。(混合デバイスは、IVS Low-Latency Streaming ユーザーガイドおよび IVS Real-Time Streaming ユーザーガイドの両方で同じです)	2025 年 7 月 28 日
Broadcast SDK: Android 1.32.2	リアルタイムストリーミングの Broadcast SDK ガイド (Android用)で、バージョン番号およびアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 7 月 25 日
同時参加者レプリケーションに制限の追加	「Service Quotas」>「その他のクオータ」で、「同時参加者レプリケーション」にクォータが追加されました。	2025 年 7 月 15 日

Broadcast SDK: Android 1.32.1、iOS 1.32.1	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 7 月 10 日
Broadcast SDK: Web 1.26.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 その他 • 「ネットワーク問題の処理」の例を更新しました。 • 「StageErrors」に FAILED エラーに関する情報を追加し、STAGE_DISCONNECTED エラーと PARTICIPANT_DISCONNECTED エラーを追加しました。	2025 年 7 月 7 日

同時パブリッシャーと同時サブスクリプションの制限を追加しました。	「Service Quotas」>「その他のクォータ」に「同時パブリッシャー」と「同時サブスクリプション」のクォータを追加しました。 「Real-Time Streaming のモニタリング」>「CloudWatch メトリクス」に ConcurrentPublishers と ConcurrentSubscriptions のメトリクスを追加しました。 「Broadcast Web SDK ガイド」の「エラー処理」>「ステージエラー」で STAGE_AT_CAPACITY テーブルエントリを更新しました。	2025 年 6 月 23 日

E-RTMP マルチトラックビデオ取り込みのサポート	「ストリーム取り込み」>「サポートされるプロトコル」に E-RTMP (拡張 RTMP) マルチトラックビデオのサポートを追加しました。 「IVS RTMP 配信」にリアルタイムストリーミング Broadcast SDK と OBS Studio を使用した E-RTMP マルチトラックビデオのストリーミングに関する情報を追加しました。 「Real-Time Streaming のモニタリング」>「CloudWatch メトリクス」にStage、Participant、SimulcastLayer、MediaType のディメンションを含む PublishFramerate メトリックを追加しました。SimulcastLayer ディメンション値も更新しました (「no-rid」と「disabled」は「none」に置き換えられました)。	2025 年 6 月 20 日
SSE-S3 暗号化	新しい情報を以下に追加しました。 • 個々の参加者の録画 – 「1. S3 バケットの作成」 • 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の開始方法」>「前提条件」	2025 年 6 月 19 日

Broadcast SDK: Web 1.25.1	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 6 月 16 日
Broadcast SDK: Web 1.25.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 6 月 12 日
Broadcast SDK: Android 1.31.0、iOS 1.31.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 6 月 12 日
ストリーム取り込み時の B フレーム	ストリーム取り込み > 対応メディア仕様で、B フレームの情報を更新しました。	2025 年 5 月 30 日

参加者のレプリケーション	この新機能の初回リリース。これらのドキュメントの変更を参照してください。 • Amazon IVS の開始方法 – 「ステップ 2: 参加者の録画オプションを使用してステージを作成する」で、コンソールの手順とスクリーンショットを更新し、個別参加者の録画を使用してステージを作成するために CLI レスポンスに `recordParticipantReplicas` を追加しました。 • 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」に、参加者のレプリケーションに関するメモを追加しました。 • EventBridge – 「例: ステージ更新」に、参加者のレプリケーション開始と参加者のレプリケーション終了の 2 つのイベントを追加しました。また、参加者配信開始、参加者配信停止、参加者配信エラーも更新されました。 • 参加者のレプリケーション – この新しいドキュメントには、概要と CLI の手順が含まれています。	2025 年 5 月 29 日

	• コスト – この新しいドキュメントには、IVS リアルタイムストリーミングに関連するコストが含まれています。 API の変更については、API リファレンス表を参照してください。	
Broadcast SDK: Android 1.30.1	リアルタイムストリーミングの Broadcast SDK ガイドのバージョン番号およびアーティファクトのリンクが更新されました (Android)。「リリースノート」を参照してください。	2025 年 5 月 26 日
Broadcast SDK: Web 1.24.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 5 月 15 日
Broadcast SDK: Android 1.30.0、iOS 1.30.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 5 月 15 日

Broadcast SDK: Web 1.23.1	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 5 月 2 日
Broadcast SDK: Web 1.23.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 また、Web Broadcast SDK ガイドの「レイヤードエンコーディングの設定 (パブリッシャー) 」に追加情報と例を追加しました。	2025 年 4 月 17 日
Broadcast SDK: Android 1.29.0、iOS 1.29.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 また、Android および iOS Broadcast SDK ガイドの「レイヤードエンコーディングの設定 (パブリッシャー)」に追加情報と例を追加しました。	2025 年 4 月 17 日

Service Quotas	「ステージあたりのコンポジション数」のクォータを追加しました。	2025 年 4 月 2 日
ストリーミングの最適化	概要には、最大ビットレート、フレームレート、解像度、デグラデーションプリファレンス (モバイルの場合) の設定に関する段落が追加されました。	2025 年 3 月 21 日
Broadcast SDK: Web 1.22.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 また、概要に別のサンプルコード例 (簡易再生) が追加されました。「補足拡張情報 (SEI) > SEI ペイロードの挿入」にメモリ使用量に関するメモが追加されました。また、「既知の問題と回避策」に `stage.leave()` に関する項目が追加されました。	2025 年 3 月 20 日
Broadcast SDK: Android 1.28.1、iOS 1.28.1	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 3 月 20 日

Broadcast SDK: Android 1.27.2、iOS 1.27.2	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 3 月 19 日
ターゲットセグメントの期間	「IVS リアルタイムストリーミングの開始」>「ステップ 2: ステージを作成する」のスクリーンショットが更新され、新しい「ターゲットセグメント期間」フィールドが表示されました。	2025 年 3 月 13 日
個々の参加者の録画	「記録を MP4 に変換」が追加されました。	2025 年 3 月 7 日

個々の参加者の記録ステッチング	この新機能の初回リリース。これらのドキュメントの変更を参照してください。 • Amazon IVS の開始方法 - 「ステップ 2: ステージを作成する」のコンソールと CLI の手順を更新しました。 • 個々の参加者の記録 – 「フラグメント化された個々の参加者の記録をマージする」を追加しました。 • EventBridge – 例: 個々の参加者の録画状態の変更に、個々の参加者の記録でマージが有効になっているときの S3 プレフィックス構築に関する情報が追加されました。	2025 年 3 月 6 日
WHIP の要件	概要テキストには、WHIP クライアントが 307 リダイレクトを処理するための要件が追加されました。	2025 年 3 月 5 日
Broadcast SDK: iOS 1.27.1	リアルタイムストリーミングの Broadcast SDK ガイドのバージョン番号およびアーティファクトのリンクが更新されました (iOS)。「リリースノート」を参照してください。	2025 年 3 月 3 日

Broadcast SDK: Web 1.21.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 2 月 20 日
Broadcast SDK: Android 1.27.0、iOS 1.27.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 2 月 20 日
Broadcast SDK: Android 1.26.0、iOS 1.26.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2025 年 1 月 30 日
Broadcast SDK: Web 1.20.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 「補足拡張情報」も更新しました。	2025 年 1 月 23 日

RTMP 配信	「RTMP エンコーダーを使用して配信する」に、ストリームにはオーディオトラックとビデオトラックの両方を含める必要があり、含まれない場合はストリームが切断されるという注意書きを記載しました。	2025 年 1 月 21 日
ネットワークの要件	この新規トップレベルページを追加しました。	2025 年 1 月 21 日
ストリーミングの最適化	「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「サイマルキャスト (パブリッシャー) によるレイヤードエンコーディングの設定」に名前を変更し、「サイマルキャスト (サブスクライバー) によるレイヤードエンコーディングの設定」を追加しました。	2024 年 12 月 12 日
Broadcast SDK: Web 1.19.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 また、「サイマルキャストによるレイヤードエンコーディング」を追加し、「イベント」に 3 つのサイマルキャスト項目を追加しました。	2024 年 12 月 12 日

Broadcast SDK: Android 1.25.0、iOS 1.25.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 各ガイドでは、さらに次のことも行っています。 • 「補足拡張情報の取得」を追加しました。 • 「サイマルキャストによるレイヤードエンコーディング」を追加しました。 • 「レンダラー」に 3 つのサイマルキャスト項目を追加しました。 • 「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」を削除しました。	2024 年 12 月 12 日
リアルタイムサムネイルの設定	「個々の参加者の録画」と「Composite Recording」で、例と JSON メタデータ情報を更新し、料金情報を追加しました。「個々の参加者の録画」に「サムネイルのみの録画」を追加しました。	2024 年 12 月 10 日

Broadcast SDK: Android 1.24.0、iOS 1.24.0	リアルタイムストリーミング Broadcast SDK ガイド (Android および iOS 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。	2024 年 11 月 13 日
サードパーティーのカメラフィルター	「IVS Broadcast SDK で Snap を使用する」の「Web」に多数の変更を加えました。	2024 年 11 月 12 日
Broadcast SDK: Web 1.18.0	低レイテンシーストリーミング Broadcast SDK ガイド (Web 用) で、バージョン番号とアーティファクトのリンクを更新しました。「リリースノート」を参照してください。 SDK ガイドに、新しいセクション「Get Supplemental Enhancement Information (SEI)」を追加しました。	2024 年 11 月 12 日
RTMP	「取り込み設定を作成する」の例を更新しました (`--ingest-protocol` を追加)。	2024 年 11 月 7 日

Broadcast SDK: Web 1.17.0	「IVS ドキュメントのランディングページ」と「低レイテンシーストリーミングの Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 10 月 10 日
Broadcast SDK: Android 1.23.0、iOS 1.23.0	「IVS ドキュメントのランディングページ」と Android 用とiOS 用のリアルタイムストリーミング Broadcast SDK ガイドでバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。 Android では、「デバッグ情報を含む SDK を使用する」を追加しました。	2024 年 10 月 10 日
Service Quotas	「参加者配信ビットレート」のクォータを追加しました。	2024 年 9 月 25 日
IVS Real-Time Streaming のモニタリング	`PublishFramerate` CloudWatch メトリクスを追加しました。	2024 年 9 月 13 日
Broadcast SDK: Web 1.16.0	「IVS ドキュメントのランディングページ」と「低レイテンシーストリーミングの Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 9 月 11 日

Broadcast SDK: Android 1.22.0、iOS 1.22.0	「IVS ドキュメントのランディングページ」と Android 用とiOS 用のリアルタイムストリーミング Broadcast SDK ガイドでバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。 また、「Android Broadcast SDK ガイド」の「開始する > ライブラリをインストールする」セクションも更新しました。	2024 年 9 月 11 日
RTMP 取り込み	IVS ストリーム取り込みページを追加しました。その下には、RTMP (新規) と WHIP の 2 つのページがあります。 「IVS Real-Time Streaming で EventBridge を使用する」に、IVS ステージ更新イベントである参加者配信エラーを追加しました。 Service Quotas では、5 つの新しい API オペレーションと 1 つの新しい IngestConfiguration クォータ (「その他のクォータ」) に TPS 値を追加しました。 API の変更については、API リファレンス表を参照してください。	2024 年 9 月 9 日

リアルタイムストリーミングの最適化	サイマルキャスト関連のさまざまな更新を行い、「レイヤーの解決」を追加しました。	2024 年 8 月 22 日
コンソール内配信/サブスクライブ	「IVS Real-Time Streaming の開始方法」で、コンソール内の配信とサブスクライブを「動画の配信とサブスクライブ」に追加しました。	2024 年 8 月 19 日
Broadcast SDK: Web 1.15.0	「IVS ドキュメントのランディングページ」と「低レイテンシーストリーミングの Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。 また、「Web Broadcast SDK ガイド」に、「参加者をサブスクライブするための設定」という新しいセクションを追加しました。 「ストリーミング最適化」に、「サブスクライバージッターバッファ MinDelay の変更」という新しいセクションを追加しました。これには、Web、Android、iOS Broadcast SDKsに関する情報が含まれます。	2024 年 8 月 15 日

Broadcast SDK: Android 1.21.0、iOS 1.21.0	「IVS ドキュメントのランディングページ」と Android 用とiOS 用のリアルタイムストリーミング Broadcast SDK ガイドでバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。 また、Android および iOS の「Broadcast SDK ガイド」に「参加者をサブスクライブするための設定」という新しいセクションを追加しました。	2024 年 8 月 15 日
録画の説明	「個々の参加者の録画 (1: S3 バケットの作成)」および「Composite Recording (前提条件、ステップ 3)」で、既存の S3 バケットの使用に関する注記を追加しました。[オブジェクト所有権] の設定は [バケット所有者に強制する] か、[バケット所有者を優先する] にする必要があります。	2024 年 8 月 13 日
Broadcast SDK: Web 1.14.0	「IVS ドキュメントのランディングページ」と「低レイテンシーストリーミングの Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 7 月 18 日

Broadcast SDK: Android 1.20.0、iOS 1.20.0	「IVS ドキュメントのランディングページ」と Android 用とiOS 用のリアルタイムストリーミング Broadcast SDK ガイドでバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 7 月 18 日
リアルタイムストリーミングの開始	「トークンスキーマ: ペイロード」と「IVS Real-Time Streaming API を使用したトークンの作成」の両方の「参加者トークンを配布する」に属性に関する情報を追加しました。	2024 年 7 月 12 日
Service Quotas	ステージサブスクライバーの最大数を 10,000 から 25,000 に増やしました。	2024 年 6 月 27 日
キーペアを使用して参加者トークンを生成する	「IVS Real-Time Streaming の開始方法」では、「参加者トークンの配布」を更新して、トークンを生成する 2 つの方法 (API とキーペア) について説明し、「キーペアでトークンを作成する」を追加しました。	2024 年 6 月 26 日

個々の参加者の録画	「録画」に関する新しいドキュメントセクションを追加し、個々の参加者の録画 (新規) と Composite Recording (既存) に関するサブドキュメントを追加しました。また、IVS での EventBridge の使用に、参加者の録画状態の変更イベントと例を追加しました。 API の変更については、API リファレンス表を参照してください。	2024 年 6 月 20 日
Service Quotas	ステージクォータを 100 から 1,000 に引き上げました。	2024 年 6 月 14 日
Broadcast SDK: Web 1.13.0	「IVS ドキュメントのランディングページ」と「低レイテンシーストリーミングの Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。 このガイドでは、新しい`ERROR`ステージイベントのエラー処理の情報を更新しました。	2024 年 6 月 13 日

Broadcast SDK: Android 1.19.0、iOS 1.19.0	「IVS ドキュメントのランディングページ」と Android 用とiOS 用のリアルタイムストリーミング Broadcast SDK ガイドでバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 6 月 13 日
Broadcast SDK: Web 1.12.0	「IVS ドキュメントのランディングページ」と「低レイテンシーストリーミングの Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。 このガイドでは、ステージ接続 ERRORED 状態に関する「ネットワークの問題の処理」に関する情報を更新しました。	2024 年 5 月 20 日
リアルタイムストリーミングの最適化	Default Layers、Qualities、Framerates では、モバイル、Low Layer の最大ビットレートを 150,000 bps から 100,000 bps に変更しました。	2024 年 5 月 16 日

Broadcast SDK: Android 1.18.0、iOS 1.18.0	「IVS ドキュメントのランディングページ」と Android 用とiOS 用のリアルタイムストリーミング Broadcast SDK ガイドでバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 5 月 16 日
Broadcast SDK: Web 1.11.0	「IVS ドキュメントのランディングページ」と「リアルタイムストリーミング Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 5 月 6 日
Broadcast SDK: Web 1.10.1	「IVS ドキュメントのランディングページ」と「リアルタイムストリーミング Broadcast SDK ガイド」の「Web」のバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 4 月 30 日
Broadcast SDK: Android 1.15.2、iOS 1.15.2	「IVS ドキュメントのランディングページ」と Android 用とiOS 用のリアルタイムストリーミング Broadcast SDK ガイドでバージョン番号とアーティファクトリンクを更新しました。「リリースノート」を参照してください。	2024 年 4 月 30 日

Broadcast SDK: iOS ガイド	「メディアストリームの配信」で、コード例を更新しました。	2024 年 4 月 26 日
Broadcast SDK: Android 1.17.0、iOS 1.17.0	リアルタイムストリーミング Broadcast SDK ガイド: Android と 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。	2024 年 4 月 22 日
サーバーサイドコンポジション	SSC では、PiP とグリッドレイアウトを説明するために、特に「Layout」でさまざまな変更を行いました。 Web Broadcast SDK ガイドに、サーバー側のレンダリングサポート が追加されました。	2024 年 3 月 26 日
OBS および WHIP サポート	OBS の WHIP で発生する可能性のある品質の問題 (断続的なビデオのフリーズなど) に関するメモを追加しました。	2024 年 3 月 22 日

Broadcast SDK: Android 1.16.0、iOS 1.16.0、および Web 1.10.0	リアルタイムストリーミング Broadcast SDK ガイド: Android、iOS、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 3 月 21 日
Broadcast SDK: Android 1.15.1、iOS 1.15.1	リアルタイムストリーミング Broadcast SDK ガイド: Android と 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。	2024 年 3 月 13 日

Broadcast SDK: モバイルオーディオモード	「オーディオモードプリセット」で、Volume Rocker プリセットカテゴリに関する情報と、Video Chat プリセットに関する iOS の既知の問題が追加されました。「アドバンストユースケース」に、誤った設定を回避する注意点を追加し、「iOS エコーキャンセレーション」と「iOS カスタムオーディオソース」のセクションを追加しました。	2024 年 3 月 1 日
Broadcast SDK: Android 1.15.0、iOS 1.15.0、および Web 1.9.0	リアルタイムストリーミング Broadcast SDK ガイド: Android、iOS、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 22 日

OBS および WHIP サポート	新しいページを追加 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 Real-Time Streaming に配信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、および Web 1.8.0	リアルタイムストリーミング Broadcast SDK ガイド: Android、iOS、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。 Android ガイドでは、新しい既知の問題 (ビデオサイズが 176x176 未満) を追加しました。 Web 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 `getDisplayMedia` 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。 リアルタイムストリーミング最適化では、サイマルキャストを使用した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りました。	2024 年 2 月 1 日

Broadcast SDK: Android 1.13.4、iOS 1.13.4、および Web 1.7.0	リアルタイムストリーミング Broadcast SDK ガイド: Android、iOS、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) メトリックの名前を DownloadPacketLoss (Stage) に変更し、IVS Real-Time Streaming 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 Real-Time Streaming 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	IVSReadOnlyAccess と IVSFullAccess の 2 つのマネージドポリシーを追加しました。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	リアルタイムストリーミング Broadcast SDK ガイド: Android と 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	リアルタイムストリーミング Broadcast 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	「参加者配信の解像度」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	リアルタイムストリーミング Broadcast SDK ガイド: Android と 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

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 Amazon EventBridge を使用する – 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 Real-Time Streaming 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。	
Composite Recording	次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 Real-Time Streaming 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
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 Real-Time Streaming 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」にディメンションのサンプル値が追加されました。	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	リアルタイムストリーミング Broadcast 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「デバイスから MediaStream 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 `ideal` のみを指定することです。 「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 Real-Time Streaming 用の CloudWatch メトリクスがリリースされました。「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	リアルタイムストリーミング Broadcast 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	リアルタイムストリーミング Broadcast 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 Real-Time Streaming 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	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: 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 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 Real-Time Streaming ドキュメントを配信しました。「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 Real-Time Streaming API リファレンスの変更

API の変更	説明	日付
StartParticipantReplication オペレーションの更新	StartParticipantReplication で、`reconnectWindowSeconds` フィールドの最大値を 60 から 300 に変更しました。	2026 年 3 月 16 日
参加者トークン交換	ExchangedParticipantToken オブジェクトを追加しました。 イベントオブジェクトに `newToken` と `previousToken` のフィールドを追加しました。 イベントオブジェクトの `name` フィールドの有効な値として `TOKEN_EXCHANGED` が追加されました。 ユーザーガイドの変更については、ユーザーガイドの表を参照してください。	2025 年 12 月 9 日

API の変更	説明	日付
SSC カスタム参加者の順序付け	GridConfiguration および PipConfiguration データ型を変更しました (`participantOrderAttribute` フィールドを追加)。これにより、Composition オブジェクトが影響を受けます。これは StartComposition リクエストおよびレスポンスと GetComposition レスポンスに影響します。 ユーザーガイドの変更については、ユーザーガイドの表を参照してください。	2025 年 9 月 16 日

API の変更	説明	日付
参加者のレプリケーション	この新機能の初回リリース。これらのドキュメントの変更を参照してください。 • 「主要なコンセプト」に参加者のレプリケーションに固有の用語を追加しました。 • ListParticipantReplicas、StartParticipantReplication、StopParticipantReplication の 3 つの新しいオペレーションを追加しました。 • ParticipantReplica という新しいオブジェクトを追加しました。 • AutoParticipantRecordingConfiguration オブジェクトに `recordParticipantReplicas` フィールドを追加しました。これは、CreateStage リクエストとレスポンス、GetStage レスポンス、UpdateStage リクエストとレスポンスに影響します。 • イベントオブジェクトに `destinationStageArn` 、`destinationSessionId` 、`replica` の 3 つのフィールドを追加しました。これにより ListParticipantEvents のレスポンスが影響を受けます。 • Participant オブジェクトと ParticipantSummary オブジェクトに `replicationState` 、`replicationType` 、`sourceStageArn` 、`sourceSessionId` の 4 つのフィールドを追加しました。これにより、GetParticipant と ListParticipants のレスポンスが影響を受けます。	2025 年 5 月 29 日

API の変更	説明	日付
	• イベントで、name フィールドに REPLICATION_STARTED 、REPLICATION_STOPPED の 2 つの有効な値を追加しました。 ユーザーガイドの変更については、ユーザーガイドの表を参照してください。	
ターゲットセグメントの期間	AutoParticipantRecordingConfiguration オブジェクトが変更され (hlsConfiguration フィールドを追加)、これによってステージオブジェクトが影響されます。これは、CreateStage リクエストとレスポンス、GetStage レスポンス、UpdateStage リクエストとレスポンスに影響します。 RecordingConfiguration オブジェクトが変更されました (hlsConfiguration フィールドを追加)。これは GetComposition レスポンスおよび StartComposition リクエストとレスポンスに影響します。 CompositionRecordingHlsConfiguration および ParticipantRecordingHlsConfiguration の 2 つのオブジェクトが追加されました。	2025 年 3 月 13 日
個々の参加者の記録ステッチング	AutoParticipantRecordingConfiguration オブジェクトが変更され (recordingReconnectWindowSeconds フィールドを追加)、これによってステージオブジェクトが影響されます。これは、CreateStage リクエストとレスポンス、GetStage レスポンス、UpdateStage リクエストとレスポンスに影響します。 Participant.recordingS3Prefix の説明が更新されました。	2025 年 3 月 6 日

API の変更	説明	日付
リアルタイムサムネイルの設定	S3DestinationConfiguration オブジェクトが変更され、`thumbnailConfigurations` が追加されました。これは GetComposition レスポンスおよび StartComposition リクエストとレスポンスに影響します。 AutoParticipantRecordingConfiguration オブジェクトが変更されました。 `thumbnailConfiguration` が追加され、 `mediaTypes` の有効な値として `NONE` が追加されました。これは、CreateStage リクエストとレスポンス、GetStage レスポンス、UpdateStage リクエストとレスポンスに影響します。 CompositionThumbnailConfiguration と ParticipantThumbnailConfiguration の 2 つのオブジェクトを追加しました。	2024 年 12 月 10 日
イベントオブジェクトとビデオオブジェクトを更新する	イベントオブジェクトに、`errorCode` の有効な値を追加しました。 Video オブジェクトでは、`height` と `width` が偶数でなければならないことを明確にしました。	2024 年 10 月 2 日

API の変更	説明	日付
RTMP 取り込み	IngestConfiguration と IngestConfigurationSummary の 2 つのオブジェクトを追加しました。IngestConfiguration エンドポイントを 5 つ追加しました (作成、削除、取得、一覧表示、更新)。 DeleteStage (オペレーションの説明) と DisconnectParticipant (オペレーションと `participantId` の説明) を更新しました。 GetParticipant レスポンスに影響する参加者オブジェクトを変更しました (`protocol` フィールドを追加)。 CreateStage、GetStage、および UpdateStage レスポンスに影響する StageEndpoints オブジェクトを変更しました (`rtmp` および `rtmps` フィールドを追加)。また、このオブジェクトの説明を更新しました (キャッシュに関する推奨事項を追加)。	2024 年 9 月 9 日
キーペアを使用して参加者トークンを生成する	3 つのオブジェクト (PublicKey、PublicKeySummary、StageEndpoints) と 4 つのエンドポイント (DeletePublicKey、GetPublicKey、ImportPublicKey、ListPublicKeys) を追加しました。CreateStage、GetStage、UpdateStage レスポンスに影響する Stage オブジェクトを変更 (`endpoints` フィールドを追加) しました。	2024 年 6 月 26 日
個々の参加者の録画	1 つのオブジェクト (AutoParticipantRecordingConfiguration) を追加し、3 つのオブジェクト (Participant、ParticipantSummary、Stage) を変更しました。これは、CreateStage リクエストとレスポンス、GetParticipant レスポンス、GetStage レスポンス、ListParticipants リクエストとレスポンス、UpdateStage リクエストとレスポンスの 5 つのエンドポイントに影響します。	2024 年 6 月 20 日

API の変更	説明	日付
ARN パターンから svs を削除する	[is]vs が指定された ARN パターンが更新され、ivs が指定されました。これは、3 つのタグエンドポイントと ChannelDestinationConfiguration$channelArn フィールドすべてに影響します。	2024 年 4 月 25 日
サーバーサイドコンポジションの更新	PipConfiguration というオブジェクトを追加しました。 2 つのオブジェクト (LayoutConfiguration、GridConfiguration) を変更しました。これは GetComposition レスポンスおよび StartComposition リクエストとレスポンスに影響します。	2024 年 3 月 13 日
Composite Recording	4 つの StorageConfiguration エンドポイントと 7 つのオブジェクト (DestinationDetail、RecordingConfiguration、S3DestinationConfiguration、S3Detail、S3StorageConfiguration、StorageConfiguration、StorageConfigurationSummary) を追加しました。 3 つのオブジェクト (Composition、Destination、DestinationConfiguration) を変更しました。これは GetComposition レスポンスおよび 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	8 つの Composition エンドポイントと EncoderConfiguration エンドポイント、および 11 のオブジェクト (ChannelDestinationConfiguration、Composition、CompositionSummary、Destination、DestinationConfiguration、DestinationSummary、EncoderConfiguration、EncoderConfigurationSummary、GridConfiguration、LayoutConfiguration、Video) を追加しました。	2023 年 11 月 16 日

API の変更	説明	日付
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 Real-Time Streaming 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 Real-Time Streaming ドキュメントを配信しました。「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 Real-Time Streaming API リファレンス」は IVS Real-Time Streaming 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

このドキュメントでは、すべての Amazon IVS Real-Time Streaming リリースノートがリリース日で整理されています。最新のリリースノートが最初に示されています。

2026 年 3 月 12 日

IVS Broadcast SDK: Web 1.33.0 (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 1.33.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • リアルタイムトークン交換の `exchangeToken` メソッドを実装しました。 • トークン交換後に `attributes` および/または `userId` が変更されたときに発生する STAGE_PARTICIPANT_METADATA_CHANGED イベントを実装しました。 • リクエストローカルステージストリーム `requestQualityStats()` メソッドに `encoderImplementation` フィールドを追加しました。

2026 年 3 月 12 日

Amazon IVS Broadcast SDK: Android 1.40.0、iOS 1.40.0 (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
Android ブロードキャスト SDK 1.40.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.40.0/android/ • TLS 証明書の検証の失敗に関するエラーメッセージを改善し、エラー列挙コードを拡張しました。
iOS Broadcast SDK 1.40.0	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/1.40.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.40.0/ios/ • TLS 証明書の検証の失敗に関するエラーメッセージを改善し、エラー列挙コードを拡張しました。

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	5.823 MB	14.139 MB
armeabi-v7a	5.046 MB	9.798 MB
x86_64	5.935 MB	14.702 MB
x86	6.190 MB	15.265 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	3.939 MB	8.013 MB

2026 年 2 月 13 日

Amazon IVS Broadcast SDK: Android 1.39.0、iOS 1.39.0 (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
Android ブロードキャスト SDK 1.39.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.39.0/android/ • STUDIO オーディオモードを使用した Bluetooth ヘッドセットの再接続に関する軽微なバグを修正。 • コア Android ビルドツールと NDK バージョンを更新しました。 • `MixedImageDevice` を停止するまれに際に発生するデッドロックを修正しました。
iOS Broadcast SDK 1.39.0	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/1.39.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.39.0/ios/ • Xcode をバージョン 26.2 に更新しました。 • このリリース以降、IVS SDK は CocoaPods 経由で配布されなくなります。

プラットフォーム	ダウンロードおよび変更
	CocoaPods は 2024 年に廃止されることが発表されており、今年後半に読み取り専用状態になります。Swift Package Manager (SPM) は、Apple がサポートする依存関係管理ソリューションとして CocoaPods に代わるもので、最新の Xcode プロジェクトで SDK を統合する標準的な方法です。 SPM に移行するか、IVS SDK フレームワークをプロジェクトに直接統合することをお勧めします。IVS SDK は、両方のアプローチで完全にサポートされています。 関連するドキュメントの変更が、次の場所で行われました。 • IVS リアルタイムストリーミングの開始 – 「ステップ 4: IVS Broadcast SDK を統合する」 > 「iOS」 • 「iOS Broadcast SDK ガイド」 – 「ライブラリのインストール」

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	5.788 MB	14.059 MB
armeabi-v7a	5.016 MB	9.740 MB
x86_64	5.898 MB	14.615 MB
x86	6.154 MB	15.184 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	3.932 MB	7.996 MB

2026 年 2 月 12 日

IVS Broadcast SDK: Web 1.32.0 (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 1.32.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • バグ修正および安定性の向上。

2026 年 1 月 13 日

Amazon IVS Broadcast SDK: Android 1.38.0、iOS 1.38.0 (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
Android ブロードキャスト SDK 1.38.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.38.0/android/ • レイヤーの優先順位付け関数 — `VERY_LOW`、`LOW`、`MEDIUM`、`HIGH`の値を持つ`Priority`列挙型が`StageVideoConfiguration.Layer` に追加されました。これにより、ネットワーク帯域幅の制約の下で最初に破棄されるレイヤーが決まります。

プラットフォーム	ダウンロードおよび変更
	• ネットワーク接続の復元後のステージ再接続の高速化。 • エラーに関連するコードを変更しました。以下の 「Mobile Broadcast SDK エラー移行ガイド」を参照してください。
iOS Broadcast SDK 1.38.0	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/1.38.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.38.0/ios/ • レイヤーの優先順位付け関数 — `VeryLow`、`Low`、`Medium`、`High`の値を持つ`IVSLocalStageStreamLayerPriority` 列挙型が追加されました。これにより、ネットワーク帯域幅の制約の下で最初に破棄されるレイヤーが決まります。 • ネットワーク接続の復元後のステージ再接続の高速化。 • エラーに関連するコードを変更しました。以下の 「Mobile Broadcast SDK エラー移行ガイド」を参照してください。

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	5.795 MB	14.070 MB
armeabi-v7a	5.021 MB	9.746 MB
x86_64	5.904 MB	14.630 MB

アーキテクチャ	圧縮サイズ	非圧縮サイズ
x86	6.161 MB	15.198 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	3.609 MB	8.078 MB

Mobile Broadcast SDK エラー移行ガイド

iOS および Android Broadcast SDK のバージョン 1.38.0 で、いくつかのエラーに関連するコードを変更しました。以前は、SDK から出力されたエラーを一意に識別するために使用できる単一のプロパティはありませんでした。代わりに、エラーの意味を理解するには、次のプロパティの組み合わせを調べる必要がありました。

Android	iOS

`BroadcastException.getCode()`
 	
`BroadcastException.getUid()`
 	
`BroadcastException.getError()`
 	
`BroadcastException.getSource()`
 	
`BroadcastException.getDetail()`
	`NSError.code`
	
	`NSError.userInfo[IVSBroadcastUidDescriptionErrorKey]`
	
	`NSError.userInfo[IVSBroadcastResultDescriptionErrorKey]`
	
	`NSError.userInfo[IVSBroadcastSourceDescriptionErrorKey]`
	
	`NSError.userInfo[NSLocalizedDescriptionKey]`

バージョン 1.38.0 以降では、`BroadcastException.getCode()` (Android) および `NSError.code` (iOS) は、パブリック `BroadcastErrorCode` (Android) および `IVSBroadcastErrorCode` (iOS) 列挙型で検索できる一意の ID を返します。

すべてのエラーに対して code を一意の ID とすることに加えて、BroadcastException.getPlatformCode() (Android用) および NSError.userInfo[IVSBroadcastPlatformCodeDescriptionErrorKey] (iOS用) フィールドが追加されました。基盤となるプラットフォームが原因でエラーが発生した場合 (ネットワークエラー、ビデオエンコードまたはデコードエラーなど)、このフィールドはゼロ以外の値になり、プラットフォームのドキュメントから追加情報を収集するために使用できます。

SDK 1.37.0 以前のバージョンからの移行

すべてのエラーを新しい戦略に準拠させるため、既存の一部エラーの値を変更する必要がありました。以下は、既存のロジックを新しいロジックにマッピングするためのガイドです。

- code が 0 以外のエラーは、コードに対して同じ値を保持します。ただし、新しい列挙定数でコードを参照すると、わかりやすくなります。たとえば、エラーを BroadcastErrorCode.Broadcast.LatencyThresholdReached と比較する方が、20401 と比較するよりも明確です。
- UID に値が設定されていた場合 (Android では -1、iOS では "-1" 以外の値だった場合) に発生したエラーについては、 code フィールドが既存の UID 値に設定されます。UID フィールドを比較する条件がある場合、定数はそのままにして、今後は code フィールドと比較できます。
- 一部のレガシーエラーには、 code または UID 値が含まれていませんでした。これらは一般的に、エラーの message (Android) または description (iOS) に基づいて照合されていましたが、エラーメッセージの動的性質のため、これはエラーを識別する信頼できる方法ではありませんでした。これらのエラーには一意に識別される特性がないため、一対一のマッピングを提供することができないためです。ただし、ほとんどのエラーは説明が変更されていないため、今後のアプリケーションリリースに向けて同じ照合ロジックを引き続き使用しながら、新しい code 値を収集して報告することが可能です。

具体的な例として、次の表のエラーチェックを次のように移行する必要があります。

Before	After
`error.code == 20401`	`error.code == BroadcastErrorCode.Broadcast.LatencyThresholdReached` 変更はありませんが、列挙値との比較を推奨します。

Before	After
`error.uid == 207`	`error.code == BroadcastErrorCode.Net.SocketRemoteHangup` code ではなく uid と比較します。
`error.message.contains("IceConnectionFailed")`	`error.code == BroadcastErrorCode.RealTime.PeerConnectionIceConnectionFailed` message (または source、または result/detail) と比較しないでください。代わりに、比較する適切な列挙型コードを見つけてください。

エラーの最も重要な部分は依然として `BroadcastException.getPlatformCode()` (Android) と `NSError.userInfo[IVSBroadcastPlatformCodeDescriptionErrorKey]` (iOS) ですが、バージョン 1.38.0 以降では、 code フィールドはエラーを一意に識別し、`BroadcastErrorCode`(Android) と `IVSBroadcastErrorCode` (iOS) の列挙型からエラー名と説明をすぐに検索できるようになります。そのため、UID、source、detail などの他のフィールドはルックアップロジックで使用しないでください。これらは補足情報としてのみ存在します。

2025 年 12 月 11 日

Amazon IVS Broadcast SDK: Android 1.37.1 (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 1.37.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.37.1/android/ • 参加者のプレビューのティアダウンに関連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	5.754 MB	13.965 MB
armeabi-v7a	4.991 MB	9.683 MB
x86_64	5.858 MB	14.529 MB
x86	6.128 MB	15.120 MB

2025 年 12 月 9 日

参加者トークン交換

参加者トークン交換の新しいサポートにより、クライアントに切断と再接続を強制することなく、IVS クライアント SDK 内のトークン機能をアップグレードまたはダウングレードし、トークン属性を更新できます。これは、参加者がサブスクライブ専用機能から開始し、後でパブリッシュ機能が必要になる共同ホスティングなどのシナリオで便利です。

「トークン交換」の新しいページを参照してください。

2025 年 12 月 5 日

IVS Broadcast SDK: Web 1.31.0 (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 1.31.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • バグ修正および安定性の向上。

2025 年 12 月 5 日

Amazon IVS Broadcast SDK: Android 1.37.0、iOS 1.37.0 (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
Android ブロードキャスト SDK 1.37.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.37.0/android/ • バグ修正および安定性の向上。 • 参加者トークン交換のサポートが追加されました。
iOS Broadcast SDK 1.37.0	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/1.37.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.37.0/ios/ • バグ修正および安定性の向上。 • 参加者トークン交換のサポートが追加されました。

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	5.753 MB	13.961 MB
armeabi-v7a	4.990 MB	9.680 MB
x86_64	5.857 MB	14.525 MB
x86	6.127 MB	15.116 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	3.588 MB	8.028 MB

2025 年 11 月 7 日

参加者ごとの録画の同期

個別参加者の録画に対応した HLS プレイリストで EXT-X-PROGRAM-DATE-TIME タグが新たにサポートされたことで、後処理時に複数の録画を正確に同期させることが可能になりました。この機能は、録画の開始時や途切れが発生した時間において、ミリ秒単位で正確な UTC タイムスタンプを提供します。これにより、参加者がネットワークの中断を経験した場合でも、同期された合成映像 (サイドバイサイドレイアウト、ピクチャーインピクチャーレイアウトなど) を作成することが可能になります。詳細については、「個々の参加者の録画」で「複数の参加者の録画を同期させる」を参照してください。

2025 年 10 月 30 日

IVS Broadcast SDK: Web 1.30.0 (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 1.30.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • バグ修正および安定性の向上。

2025 年 10 月 30 日

Amazon IVS Broadcast SDK: Android 1.36.0、iOS 1.36.0 (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
Android ブロードキャスト SDK 1.36.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.36.0/android/ • 長時間にわたってバックグラウンドにあったカメラがフォアグラウンドに戻るときのカメラの復旧を改善しました。 • `ImageDevice` に `embedMessage` メソッドを追加して、配信中のビデオストリームにメタデータペイロードを挿入できるようにしました。「Android Broadcast SDK ガイド」の「メッセージを埋め込む」を参照してください。
iOS Broadcast SDK 1.36.0	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/1.36.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.36.0/ios/ • `IVSImageDevice` に `embedMessage` メソッドを追加して、配信中のビデオストリームにメタデータペイロードを挿入できるようにしました。「iOS Broadcast SDK ガイド」の「メッセージを埋め込む」を参照してください。

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	5.736 MB	13.898 MB
armeabi-v7a	4.974 MB	9.638 MB
x86_64	5.839 MB	14.456 MB
x86	6.109 MB	15.047 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	3.569 MB	7.962 MB

2025 年 10 月 14 日

リアルタイム制限: コンポジションを更新しました

「アカウントあたりの同時 Composition リソースの最大数」のクォータを 5 から 20 に更新しました。「Service Quotas」 > 「その他のクオータ」に記載されています。

2025 年 10 月 2 日

IVS Broadcast SDK: Web 1.29.0 (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 1.29.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • バグ修正および安定性の向上。

2025 年 10 月 2 日

Amazon IVS Broadcast SDK: Android 1.35.0、iOS 1.35.0 (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
Android ブロードキャスト SDK 1.35.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.35.0/android/ • バグ修正および安定性の向上。
iOS Broadcast SDK 1.35.0	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/1.35.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.35.0/ios/ • `IVSImageDevice.setOnFrameCallback` は `DispatchQueue` でカスタマイズ可能であり、オプションでフレームに関連付けられた `CVPixelBuffer` を含めることができます。

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	5.730 MB	13.900 MB
armeabi-v7a	4.971 MB	9.639 MB
x86_64	5.835 MB	14.455 MB
x86	6.104 MB	15.041 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	3.569 MB	7.963 MB

2025 年 9 月 16 日

サーバーサイドコンポジションのカスタム参加者の順序付け

SSC のカスタム参加者順序の新たなサポートにより、グリッドレイアウトと Picture-in-Picture (PiP) レイアウトの両方で参加者の配置をきめ細かく制御できます。「サーバーサイドコンポジション」(`participantOrderAttribute` および「カスタム参加者順序」の追加を含むさまざまな変更) および「IVS Real-Time Streaming API リファレンス」(Composition オブジェクトに `participantOrderAttribute` を追加) を参照してください。

2025 年 9 月 11 日

Amazon IVS Broadcast SDK: Android 1.34.0、iOS 1.34.0 (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
Android ブロードキャスト SDK 1.34.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.34.0/android/ • メディアトランスポートを配信およびサブスクライブするための CPU の改善。 • `packetsLost` が`LocalVideoStats` と`LocalAudioStats` に追加されました。
iOS Broadcast SDK 1.34.0	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/1.34.0/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/1.34.0/ios/ • メディアトランスポートを配信およびサブスクライブするための CPU の改善。 • `packetsLost` が`IVSLocalVideoStats` と`IVSLocalAudioStats` に追加されました。 • ステージを離れた後にデバイスがデタッチされないバグを修正しました。プライバシーインジケータが予期せず点灯する可能性がありました。

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	5.796 MB	14.089 MB
armeabi-v7a	5.036 MB	9.788 MB
x86_64	5.906 MB	14.653 MB
x86	6.174 MB	15.240 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	3.594 MB	8.046 MB

2025 年 9 月 10 日

インターフェイス VPC エンドポイント

インターフェイス VPC (Virtual Private Cloud) エンドポイントの新しいサポートにより、安全なライブビデオ取り込みを必要とするワークロード向けに、Amazon VPC と IVS 間の安全なプライベート接続を確立できます。これにより、IVS の取り込みトラフィックは AWS ネットワーク内にとどまり、パブリックインターネットを経由することはありません。インターフェイス VPC エンドポイントは AWS PrivateLink を使用しています。これは、Amazon VPC で Elastic Network Interface とプライベート IP を使用して AWS のサービス間のプライベート通信を可能にする AWS のテクノロジーです。「IVS 低レイテンシーストリーミングユーザーガイド」の「Private Ingest」および「IVSリアルタイムストリーミングユーザーガイド」の「Private Ingest to Stages」を参照してください。

2025 年 9 月 4 日

IVS Broadcast SDK: Web 1.28.0 (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 1.28.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 削除されたステージに参加するか、切断された参加者トークンを使用してステージに参加すると、`TIMEOUT` の代わりに `STAGE_DELETED` または `STAGE_DISCONNECTED` エラーが報告されるようになりました。 • サイマルキャストに関連する内部ポーリングリクエストを最適化しました。

2025 年 8 月 7 日

IVS Broadcast SDK: Web 1.27.0 (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 1.27.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • `requestRTCStats` から取得されたビデオおよびオーディオの統計の簡易オブジェクトを公開する `requestQualityStats` が `RemoteStageStream` に追加されました。 • `RemoteStageStream` のミュート状態およびその `mediaStreamTrack` 有効状態が常に同期されるように更新されます。

2025 年 8 月 7 日

Amazon IVS Broadcast SDK: Android 1.33.0、iOS 1.33.0 (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

Android Broadcast SDK 1.33.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.33.0/android/
	• デバイストーチを制御する新しい方法
	  • `CameraSource.Capabilities` は `isTorchSupported` を実装します。
	  • `CameraSource.Options.Builder` は `setEnableTorch` を実装します。
	• Android Broadcast SDK は、Google Play の「16 KB のページサイズの互換性要件」

プラットフォーム	ダウンロードおよび変更
	を満たしています。(注: SDK のバージョン 1.23.0 の時点で実装されています)
iOS Broadcast SDK 1.33.0	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/1.33.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.33.0/ios/ • デバイストーチを制御する新しい方法: `IVSImageDevice` は `isTorchSupported` および `torchEnabled` の2つのプロパティを実装します。デバイスが `isTorchSupported` でトーチをサポートしているかどうかを確認し、`torchEnabled` を設定して切り替えます。 • ピア接続のタイムアウトが生じる可能性のある iOS 18.5 以降の特定の VPN の問題を解決しました。(注: SDK のバージョン 1.32.1 の時点で実装されています)

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	5.689 MB	13.829 MB
armeabi-v7a	4.962 MB	9.649 MB
x86_64	5.806 MB	14.413 MB
x86	6.066 MB	14.983 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	3.505 MB	7.828 MB

2025 年 7 月 25 日

Amazon IVS Broadcast SDK: Android 1.32.2 (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
Android Broadcast SDK 1.32.2	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.32.2/android/ • Stage 接続の IPv6 が無効になりました。

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	5.693 MB	13.838 MB
armeabi-v7a	4.964 MB	9.653 MB
x86_64	5.810 MB	14.422 MB
x86	6.067 MB	14.988 MB

2025 年 7 月 23 日

新しいリアルタイムメトリクスと制限の実施: 同時パブリッシャーとサブスクリプション

6 月 23 日、AWS リージョンのすべてのステージに同時パブリッシャーおよび同時サブスクリプションの最大数に対し、調整可能な新しい Service Quotas が 2 つ導入されました。本日、これらの新しいクォータの実施が開始されます。

2025 年 7 月 15 日

新しいリアルタイム制限: 同時参加者レプリケーション

AWS リージョンのすべてのステージで参加者あたりの同時レプリケーションの最大数に対し、調整不可能な新しい Service Quotas が導入されました。「Service Quotas」 > 「その他のクオータ」に記載されています。

2025 年 7 月 10 日

Amazon IVS Broadcast SDK: Android 1.32.1、iOS 1.32.1 (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 1.32.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.32.1/android/ • `StageAudioConfiguration.enableEchoCancellation()` を削除しました。代わりに、`StageAudioManager` を使用してエコーキャンセレーションを有効または無効にします。 • エコーキャンセレーションを無効にするために、`StageAudioManager` の `STUDIO` と `SUBSCRIBE_ONLY` のプリセットを変更し

プラットフォーム	ダウンロードおよび変更
	ました。STUDIO でエコーキャンセレーションを使用する場合は、最初にプリセットを設定し、次にエコーキャンセレーションを有効にして、エコーキャンセルなしの STUDIO のデフォルト設定を上書きします。 • ステージにさらに複雑なオーディオとビジュアルを配信するために使用できる単一の出力 Device に複数の画像とオーディオソースを合成するための MixedDevice API スイートを追加しました。
iOS Broadcast SDK 1.32.1	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/1.32.1/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.32.1/ios/ • ステージにさらに複雑なオーディオとビジュアルを配信するために使用できる単一の出力 IVSDevice に複数の画像とオーディオソースを合成するための IVSMixedDevice API スイートを追加しました。

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	5.692 MB	13.840 MB
armeabi-v7a	4.965 MB	9.655 MB
x86_64	5.810 MB	14.424 MB
x86	6.068 MB	14.990 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	3.508 MB	7.900 MB

2025 年 7 月 7 日

IVS Broadcast SDK: Web 1.26.0 (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 1.26.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • requestRTCStats から取得されたビデオとオーディオの統計の簡易オブジェクトを公開する `requestQualityStats` が LocalStageStream に追加されました。 • セットアップ中に発生して後続の結合の失敗の原因になる可能性のある WebSocket リークを修正しました。 • 失敗したサブスクリプションまたは配信オペレーションを再試行するときに 1302 エラーが誤って表示される問題を修正しました。 • 結合接続が ERRORED または CONNECTING 状態の場合、サブスクリプションと配信接続の再試行の安定性が向上しました。

2025 年 6 月 23 日

新しいリアルタイムメトリクスと制限: 同時パブリッシャーとサブスクリプション

AWS リージョンのすべてのステージにわたる同時パブリッシャーと同時サブスクリプションの最大数で、2 つの新しい調整可能な Service Quotas が導入されました。詳細については、「Service Quotas」>「その他のクォータ」を参照してください。これらのクォータにより、アカウント全体の総使用量をより細かく制御できます。以前は、IVS ではステージあたりのパブリッシャーとサブスクライバーの数にのみ制限が適用されていました。この設定では、アカウントレベルで保護を設定することが難しくなり、多くのステージを作成するお客様の場合、予想よりも使用量や関連コストが高くなる可能性がありました。

注: これらの新しいクォータの適用は 7 月 23 日に開始されます。30 日の間、使用量を確認して、必要に応じて Service Quotas の引き上げをリクエストできます。

また、2 つの新しい CloudWatch メトリクス (`ConcurrentPublishers` と `ConcurrentSubscriptions`) が追加されました。これらのメトリクスは、すべてのステージにわたって使用状況をモニタリングし、デフォルトの制限に近づいているかどうかを評価するのに役立ちます。詳細については、「Real-Time Streaming のモニタリング」>「CloudWatch メトリクス」を参照してください。使用量がクォータ制限に近づいたときに警告するように CloudWatch アラームを設定することをお勧めします。

2025 年 6 月 20 日

E-RTMP マルチトラックビデオ取り込みのサポート

E-RTMP (拡張リアルタイムメッセージングプロトコル) マルチトラックビデオを使用して、複数の動画品質を IVS ステージに送信できます。この機能により、アダプティブビットレートストリーミングが可能になり、視聴者はネットワーク接続に最適な品質で視聴できます。IVS RTMP Publishing ドキュメントの「E-RTMP Multitrack Video」を参照してください。

2025 年 6 月 16 日

IVS Broadcast SDK: Web 1.25.1 (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
Web Broadcast SDK 1.25.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • v22 の NPM の意図しないエンジンの適用を削除しました。パッケージがトランスパイルされるため、すべての LTS ノードバージョンがサポートされています。

2025 年 6 月 12 日

Amazon IVS Broadcast SDK: Android 1.31.0、iOS 1.31.0 (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
Android Broadcast SDK 1.31.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.31.0/android/ • バグ修正および安定性の向上。
iOS Broadcast SDK 1.31.0	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/1.31.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.31.0/ios/ • バグ修正および安定性の向上。

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	5.579 MB	13.594 MB
armeabi-v7a	4.864 MB	9.473 MB
x86_64	5.697 MB	14.173 MB
x86	5.951 MB	14.724 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	3.431 MB	7.732 MB

2025 年 6 月 12 日

IVS Broadcast SDK: Web 1.25.0 (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 1.25.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • リモート参加者が ERROR 状態になった後に SEI メッセージの送信に失敗する可能性があったバグを修正しました。 • STAGE_STREAM_MUTE_CHANGED ステージイベントが呼び出された際に、複数のリモートステージストリームが返される可能性があったバグを修正しました。

プラットフォーム	ダウンロードおよび変更
	• エラーが発生したストリームに対して STAGE_PARTICIPANT_STREAMS_REMOVED が呼び出されないバグを修正しました。

2025 年 5 月 29 日

参加者のレプリケーション

参加者のレプリケーションでは、あるステージから別のステージに参加者をコピーできます。これは、同じ参加者を複数のステージに同時に表示し、ステージ間のインタラクションを有効にする場合に便利です。ドキュメントの変更については、「ドキュメント履歴」(ユーザーガイドと API リファレンステーブルの両方) を参照してください。

2025 年 5 月 26 日

Amazon IVS Broadcast SDK: Android 1.30.1 (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 1.30.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.30.1/android/ • 一部の Android デバイスで、DeviceDiscovery を使用して SDK 管理のマイクを STUDIO オーディオプリセットで使用した際に、マイクの音量が小さくなるバグを修正しました。

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	5.579 MB	13.592 MB
armeabi-v7a	4.863 MB	9.472 MB
x86_64	5.696 MB	14.171 MB
x86	5.950 MB	14.722 MB

2025 年 5 月 15 日

IVS Broadcast SDK: Web 1.24.0 (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 1.24.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • ステージを退出して再参加するときのメモリリークを修正しました。

2025 年 5 月 15 日

Amazon IVS Broadcast SDK: Android 1.30.0、iOS 1.30.0 (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
Android Broadcast SDK 1.30.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.30.0/android/ • バグ修正および安定性の向上。

プラットフォーム	ダウンロードおよび変更
iOS Broadcast SDK 1.30.0	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/1.30.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.30.0/ios/ • バグ修正および安定性の向上。

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	5.571 MB	13.577 MB
armeabi-v7a	4.857 MB	9.462 MB
x86_64	5.691 MB	14.156 MB
x86	5.944 MB	14.708 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	3.430 MB	7.732 MB

2025 年 5 月 2 日

IVS Broadcast SDK: Web 1.23.1 (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
Web Broadcast SDK 1.23.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • `join()` の解決前に、参加型イベントが常に発生する問題を修正しました。 • 短時間で退出して再参加した際に、ローカル参加者が誤ってリモート参加者として報告される問題を修正しました。

2025 年 4 月 17 日

Amazon IVS Broadcast SDK: Android 1.29.0、iOS 1.29.0 (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
Android Broadcast SDK 1.29.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.29.0/android/ • サイマルキャストパブリッシャーコントロール機能が追加されました。Android Broadcast SDK ガイドの「レイヤードエンコーディングの設定 (パブリッシャー) 」を参照してください。 • バグ修正および安定性の向上。
iOS Broadcast SDK 1.29.0	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/1.29.0/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/1.29.0/ios/ • サイマルキャストパブリッシャーコントロール機能が追加されました。iOS Broadcast SDK ガイドの「レイヤードエンコーディングの設定 (パブリッシャー) 」を参照してください。 • バグ修正および安定性の向上。

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	5.566 MB	13.546 MB
armeabi-v7a	4.853 MB	9.444 MB
x86_64	5.681 MB	14.119 MB
x86	5.939 MB	14.674 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	3.429 MB	7.715 MB

2025 年 4 月 17 日

IVS Broadcast SDK: Web 1.23.0 (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 1.23.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • サイマルキャストパブリッシャーコントロール機能が追加されました。Web Broadcast SDK Guideの「レイヤードエンコーディングの設定 (パブリッシャー) 」を参照してください。 • 配信レイテンシーを改善しました。これは PUBLISHED イベントのタイミングに影響します。 • ステージへの接続が失われたが回復可能 (特に、JOIN_ERROR カテゴリの FAILED および TIMEOUT エラー) である可能性があるときに、SDK が ERROR コールバックを介して join カテゴリーエラーを発生させるバグを修正しました。 • insertSeiMessage オペレーションのバグを修正しました。戦略の更新により、後続の insertSeiMessage の呼び出しで SEI メッセージを送信できない可能性があります。

2025 年 4 月 2 日

新しいクォータ: ステージあたりのコンポジション数

ステージごとに許可される同時コンポジションの最大数に新しいクォータを追加しました。これは、Service Quotas > Other Quotas に記載されています。

2025 年 3 月 20 日

Amazon IVS Broadcast SDK: Android 1.28.1、iOS 1.28.1 (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 1.28.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.28.1/android/ • バグ修正および安定性の向上。
iOS Broadcast SDK 1.28.1	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/1.28.1/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.28.1/ios/ • バグ修正および安定性の向上。

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	5.613 MB	13.760 MB
armeabi-v7a	4.885 MB	9.558 MB
x86_64	5.728 MB	14.342 MB

アーキテクチャ	圧縮サイズ	非圧縮サイズ
x86	5.987 MB	14.923 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	3.417 MB	7.698 MB

2025 年 3 月 20 日

IVS Broadcast SDK: Web 1.22.0 (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 1.22.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • preferredLayerForStream 戦略メソッドに有効な戻り型として null が追加されました。 • ストリームの開始後に新しいレイヤーが利用可能になった際に preferredLayerForStream が再度呼び出されないバグが修正されました。 • ストリームの開始後に stream.getHighestQualityLayer が最高品質のレイヤーを選択しないバグが修正されました。

2025 年 3 月 19 日

Amazon IVS Broadcast SDK: Android 1.27.2、iOS 1.27.2 (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
Android Broadcast SDK 1.27.2	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.27.2/android/ • 50 以上のステージを作成する際に一部のデバイスに影響を与えるリソースリークのリグレッションが修正されました。 • サードパーティーの配信ソフトウェアを使用する際にビデオのフリーズ率が高くなる可能性があるリグレッションを修正しました。
iOS Broadcast SDK 1.27.2	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/1.27.2/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.27.2/ios/ • サードパーティーの配信ソフトウェアを使用する際にビデオのフリーズ率が高くなる可能性があるリグレッションを修正しました。

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	5.700 MB	14.197 MB
armeabi-v7a	4.945 MB	9.879 MB
x86_64	5.810 MB	14.802 MB

アーキテクチャ	圧縮サイズ	非圧縮サイズ
x86	6.073 MB	15.412 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	3.622 MB	8.584 MB

2025 年 3 月 13 日

ターゲットセグメントの期間

このリリースは IVS リアルタイムストリーミング API に追加され、複合記録またはステージ参加者の記録のいずれかを使用して生成された記録セグメントのターゲット期間を定義できるようになります。特定の API 変更については、「ドキュメント履歴」(ユーザーガイドおよび API リファレンステーブルの両方) を参照してください。

2025 年 3 月 6 日

個々の参加者の記録ステッチング

これは、新しい機能の最初のリリースです。ステージが個々の参加者の記録用に設定されている場合、ステージパブリッシャーがステージから切断してから再接続した場合に IVS が前のセッションと同じ S3 プレフィックスに記録を試みる際、時間枠を指定できるようになりました。つまり、パブリッシャーが切断してから指定された時間内で再接続した場合、複数の記録は 1 つの記録として見なされてマージされます。ドキュメントの変更については、「ドキュメント履歴」(ユーザーガイドと API リファレンステーブルの両方) を参照してください。

2025 年 3 月 3 日

Amazon IVS Broadcast SDK: Web 1.27.1 (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
iOS Broadcast SDK 1.27.1	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/1.27.1/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.27.1/ios/ • Pro デバイスで超ワイドレンズを使用しながらカメラの近くに置かれている物体の焦点パフォーマンスが改善されました。

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	3.625 MB	8.601 MB

2025 年 2 月 20 日

Amazon IVS Broadcast SDK: Android 1.27.0、iOS 1.27.0 (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
Android Broadcast SDK 1.27.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.27.0/android/ • バグ修正および安定性の向上。

プラットフォーム	ダウンロードおよび変更
iOS Broadcast SDK 1.27.0	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/1.27.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.27.0/ios/ • バグ修正および安定性の向上。

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	5.700 MB	14.197 MB
armeabi-v7a	4.944 MB	9.879 MB
x86_64	5.809 MB	14.802 MB
x86	6.073 MB	15.412 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	3.625 MB	8.601 MB

2025 年 2 月 20 日

IVS Broadcast SDK: Web 1.21.0 (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 1.21.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 有効な戻り値として `null` を含むように、`preferredLayerForStream` の戦略タイプを更新しました。 • TSconfig `skipLibCheck` が false に設定されている場合の TypeScript コンパイルエラーを修正しました。 注: このリリースの一環として、型が 1 つのロールアップに統合されました。アプリケーションがパスに基づいてネストされた型をインポートする場合、エラーが発生する可能性があります。エラーが発生する場合は、インポートを単純に `'amazon-ivs-broadcast'` に変更します。

2025 年 1 月 30 日

Amazon IVS Broadcast SDK: Android 1.26.0、iOS 1.26.0 (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
Android Broadcast SDK 1.26.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.26.0/android/

プラットフォーム	ダウンロードおよび変更
	• バグ修正および安定性の向上。
iOS Broadcast SDK 1.26.0	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/1.26.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.26.0/ios/ • バグ修正および安定性の向上。

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	5.695 MB	14.186 MB
armeabi-v7a	4.939 MB	9.872 MB
x86_64	5.804 MB	14.790 MB
x86	6.065 MB	15.398 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	3.624 MB	8.601 MB

2025 年 1 月 23 日

IVS Broadcast SDK: Web 1.20.0 (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 1.20.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • LocalStageStream に `insertSeiMessage` メソッドを追加して、ビデオ配信ストリームに補足拡張情報 (SEI) ペイロードを挿入できるようにしました。「IVS Broadcast SDK: Web ガイド」の「補足拡張情報」を参照してください。

2024 年 12 月 12 日

Amazon IVS Broadcast SDK: Android 1.25.0、iOS 1.25.0 (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
Android Broadcast SDK 1.25.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.25.0/android/ • サイマルキャストコントロール機能が追加されました。「ストリーミングの最適化」の「サイマルキャスト (サブスクライバー) によるレイヤードエンコーディングの設定」を参照してください。 • ImageDeviceFrame オブジェクトで新しいフィールドを持つサブスクライバーが、SEI (補足拡張情報) ペイロードを利用できるよ

プラットフォーム	ダウンロードおよび変更
	うになりました。「IVS Broadcast SDK: Android ガイド」の「補足拡張情報 (SEI) の取得」を参照してください。 • 受信オーディオストリームの初期ゲイン値の設定を許可するため、`SubscribeConfiguration::setInitialGain` メソッドを追加しました。 • バグ修正および安定性の向上。
iOS Broadcast SDK 1.25.0	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/1.25.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.25.0/ios/ • サイマルキャストコントロール機能が追加されました。「ストリーミングの最適化」の「サイマルキャスト (サブスクライバー) によるレイヤードエンコーディングの設定」を参照してください。 • IVSImageDeviceFrame オブジェクトで新しいフィールドを持つサブスクライバーが、SEI (補足拡張情報) ペイロードを利用できるようになりました。「IVS Broadcast SDK: iOS ガイド」の「補足拡張情報 (SEI) の取得」を参照してください。 • 受信オーディオストリームの初期ゲイン値の設定を許可するため、`IVSSubscribeConfiguration.initialGain` メソッドを追加しました。 • バグ修正および安定性の向上。

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	5.677 MB	14.103 MB
armeabi-v7a	4.905 MB	9.791 MB
x86_64	5.786 MB	14.725 MB
x86	6.030 MB	15.302 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	3.625 MB	8.585 MB

2024 年 12 月 12 日

IVS Broadcast SDK: Web 1.19.0 (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 1.19.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • サイマルキャストコントロール機能が追加されました。「ストリーミングの最適化」の「サイマルキャスト (サブスクライバー) によるレイヤードエンコーディングの設定」を参照してください。 • バグ修正および安定性の向上。

2024 年 12 月 10 日

リアルタイムストリーミングサムネイルの設定

このリリースでは、ライブセッションのサムネイルの記録を有効/無効にし、ライブセッションのサムネイルが生成される間隔を変更できます。これは、この新しい機能の最初のリリースです。以下を参照してください。

- 個々の参加者の録画 – 例と JSON メタデータ情報を更新し、料金情報と「サムネイルのみの録画」を追加しました。
- Composite Recording – サンプルと JSON メタデータ情報を更新し、料金情報を追加しました。
- API リファレンス RT – いくつかの変更を行いました。
  - S3DestinationConfiguration オブジェクトが変更され、`thumbnailConfigurations` が追加されました。これは GetComposition レスポンスおよび StartComposition リクエストとレスポンスに影響します。
  - AutoParticipantRecordingConfiguration オブジェクトが変更されました。`thumbnailConfiguration` が追加され、 `mediaTypes` の有効な値として `NONE` が追加されました。これは、CreateStage リクエストとレスポンス、GetStage レスポンス、UpdateStage リクエストとレスポンスに影響します。
  - CompositionThumbnailConfiguration と ParticipantThumbnailConfiguration の 2 つのオブジェクトが追加されました。

2024 年 11 月 13 日

Amazon IVS Broadcast SDK: Android 1.24.0、iOS 1.24.0 (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
Android Broadcast SDK 1.24.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.24.0/android/ • バグ修正と安定性の向上。

プラットフォーム	ダウンロードおよび変更
iOS Broadcast SDK 1.24.0	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/1.24.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.24.0/ios/ • バグ修正と安定性の向上。

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	5.521 MB	13.791 MB
armeabi-v7a	4.789 MB	9.623 MB
x86_64	5.718 MB	14.709 MB
x86	5.933 MB	15.163 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	3.589 MB	8.466 MB

2024 年 11 月 12 日

IVS Broadcast SDK: Web 1.18.0 (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 1.18.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • サブスクライバーが SEI (補足拡張情報) ペイロードを利用できるようにするための新しいイベントを追加しました。 • 配信停止リクエストおよびサブスクライブ解除リクエスト中に発生する例外を修正しました。 • 急速に参加したり、退出したりすると、他の参加者にエラーが発生する競合状態を修正しました。

2024 年 10 月 10 日

IVS Broadcast SDK: Web 1.17.0 (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 1.17.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 軽微なバグを修正。

2024 年 10 月 10 日

Amazon IVS Broadcast SDK: Android 1.23.0、iOS 1.23.0 (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
Android Broadcast SDK 1.23.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/android/ • このリリースでは、デバッグ情報を含む Android Broadcast SDK のバージョンの公開も開始しました。「デバッグ情報を含む SDK の使用」を参照してください。 • 軽微なバグを修正。
iOS Broadcast SDK 1.23.0	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/1.23.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/ios/ • 軽微なバグを修正。

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	5.432 MB	13.560 MB
armeabi-v7a	4.707 MB	9.451 MB
x86_64	5.626 MB	14.459 MB
x86	5.838 MB	14.908 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	3.542 MB	8.316 MB

2024 年 9 月 11 日

Amazon IVS Broadcast SDK: Android 1.22.0、iOS 1.22.0 (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
Android Broadcast SDK 1.22.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.22.0/android/ • カメラ入力の切り替え後に、特定の Android デバイスがプレビューに黒いフレームを表示するバグを修正しました。 • 軽微なバグを修正。
iOS Broadcast SDK 1.22.0	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/1.22.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.22.0/ios/ • 軽微なバグを修正。

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	5.359 MB	13.392 MB

アーキテクチャ	圧縮サイズ	非圧縮サイズ
armeabi-v7a	4.636 MB	9.325 MB
x86_64	5.548 MB	14.268 MB
x86	5.754 MB	14.710 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	3.488 MB	8.199 MB

2024 年 9 月 11 日

IVS Broadcast SDK: Web 1.16.0 (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 1.16.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 軽微なバグを修正。

2024 年 9 月 9 日

RTMP 取り込み

IVS Broadcast SDK を使用する代わりに、(既にサポートされている WHIP に加えて) RTMP ソースから IVS ステージに動画を配信できるようになりました。ドキュメントの変更については、「ドキュメント履歴」(ユーザーガイドと API リファレンステーブルの両方) を参照してください。

2024 年 8 月 19 日

コンソール内配信/サブスクライブ

IVS コンソールから配信およびサブスクライブできるようになりました。「IVS Real-Time Streaming の開始」の「動画の配信とサブスクライブ」を参照してください。

2024 年 8 月 15 日

IVS Broadcast SDK: Web 1.15.0 (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 1.15.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • `join()` が繰り返し呼び出されるときにパブリッシャーメディアの品質に影響するレース条件を修正しました。連続して `join()` を呼び出すと、配信状態とストリーム状態の変更とともに、`STAGE_PARTICIPANT_JOINED` イベントが再トリガーされなくなります。 • トークン `attributes` フィールドでテキスト以外の文字を使用した場合に、参加者トークンの解析で問題が発生するバグを修正しました。 • 参加者のサブスクライバーを設定する方法を追加しました。最初は、ジッターバッファの最小遅延のみを設定できます。SDK リファレンスドキュメント、「ウェブブロードキャストWeb Broadcast SDK ガイド」の「参加者へのサブスクライブの設定」、「ストリーミング最適化」の「サブスクライバージッ

プラットフォーム	ダウンロードおよび変更
	ターバッファ MinDelay の変更」を参照してください。

2024 年 8 月 15 日

Amazon IVS Broadcast SDK: Android 1.21.0、iOS 1.21.0 (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
Android Broadcast SDK 1.21.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.21.0/android/ • MT6765 チップセットを使用するバグに影響するデバイスを修正しました。サブスクライバープレビューでは、状況によってはブラックフレームがレンダリングされます。 • 参加者のサブスクライバーを設定する方法を追加しました。最初は、ジッターバッファの最小遅延のみを設定できます。SDK リファレンスドキュメント、「Android Broadcast SDK ガイド」の「参加者へのサブスクライブの設定」、「ストリーミング最適化」の「サブスクライバージッターバッファ MinDelay の変更」を参照してください。 • 軽微なバグを修正。
iOS Broadcast SDK 1.21.0	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/1.21.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.21.0/ios/

プラットフォーム	ダウンロードおよび変更
	• 参加者のサブスクライバーを設定する方法を追加しました。最初は、ジッターバッファの最小遅延のみを設定できます。SDK リファレンスドキュメント、「iOS Broadcast SDK ガイド」の「参加者へのサブスクライブの設定」、「ストリーミング最適化」の「サブスクライバージッターバッファ MinDelay の変更」を参照してください。 • 軽微なバグを修正。

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	5.350 MB	13.378 MB
armeabi-v7a	4.628 MB	9.312 MB
x86_64	5.538 MB	14.253 MB
x86	5.744 MB	14.694 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	3.485 MB	8.199 MB

2024 年 7 月 18 日

IVS Broadcast SDK: Web 1.14.0 (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 1.14.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • API ドキュメントの改善 • 接続のリセット中に報告されたビデオおよびオーディオ統計の外れ値を修正しました。 • 軽微な依存関係の更新。

2024 年 7 月 18 日

Amazon IVS Broadcast SDK: Android 1.20.0、iOS 1.20.0 (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
Android Broadcast SDK 1.20.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.20.0/android • Intel プロセッサーを搭載した Chromebooks で Broadcast SDK を実行できないバグを修正しました。 • 軽微なバグを修正。
iOS Broadcast SDK 1.20.0	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/1.20.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.20.0/ios

プラットフォーム	ダウンロードおよび変更
	• 軽微なバグを修正。

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	5.318 MB	13.299 MB
armeabi-v7a	4.605 MB	9.254 MB
x86_64	5.507 MB	14.168 MB
x86	5.715 MB	14.608 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	3.465 MB	8.164 MB

2024 年 6 月 26 日

キーペアを使用して参加者トークンを生成する

キーペアを使用して、独自のサーバーアプリケーションで参加者トークンを生成できるようになりました。これにより、参加者トークンが必要なたびに CreateParticipantToken を呼び出す必要がなくなります。ドキュメントの変更については、「ドキュメント履歴」(ユーザーガイドと API リファレンステーブルの両方) を参照してください。

2024 年 6 月 20 日

個々の参加者の録画

個々の参加者の録画により、IVS Real-Time Streaming のお客様は IVS ステージパブリッシャーを個別に S3 バケットに録画できます。「録画」、「個々の参加者の録画」、および「リアルタイムストリーミング API リファレンス」の変更を参照してください。(特定のドキュメントの変更については、「ドキュメント履歴」を参照してください。)

2024 年 6 月 13 日

Amazon IVS Broadcast SDK: Android 1.19.0、iOS 1.19.0 (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
Android Broadcast SDK 1.19.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.19.0/android • 最近の Android バージョンでは、画面をキャプチャするときに表示される通知にアイコンが必要です。必要に応じて、`Session # createServiceNotificationBuilder` によって返された `Notification.Builder` で、`setSmallIcon` を呼び出すことで、アイコンをカスタマイズできるようになりました。 • Wifi からセルラー接続に移行するデバイスの接続復旧時間を改善しました。これには、`CHANGE_NETWORK_STATE` アクセス許可が必要です。
iOS Broadcast SDK 1.19.0	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/1.19.0/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/1.19.0/ios • 軽微なバグを修正。

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	5.304 MB	13.340 MB
armeabi-v7a	4.598 MB	9.299 MB
x86_64	5.495 MB	14.207 MB
x86	5.694 MB	14.625 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	3.393 MB	7.949 MB

2024 年 6 月 13 日

IVS Broadcast SDK: Web 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
Web Broadcast SDK 1.13.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference

プラットフォーム	ダウンロードおよび変更
	• `StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED` および `StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` のイベント変更動作の期間を更新しました。参加者は、`ERRORED` イベントが終了するまで、`ATTEMPTING_SUBSCRIBE` または `ATTEMPTING_PUBLISH` の状態に留まれるようになりました。 • SDK で発生したエラーをリッスンするための `StageEvents.ERROR` イベントを追加しました。詳細については、「リアルタイム Broadcast SDK: Web ガイド」の「エラー処理」を参照してください。

2024 年 5 月 20 日

IVS Broadcast SDK: Web 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 1.12.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 配信およびサブスクライブオペレーションの再試行処理が改善されました。 • 分析、特にレイテンシーとオーディオ品質の測定が改善されました。

2024 年 5 月 16 日

Amazon IVS Broadcast SDK: Android 1.18.0、iOS 1.18.0 (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
Android Broadcast SDK 1.18.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.18.0/android • SDK は、接続されたステージが AWS コントロールプレーンによって削除されたとき、または使用中のトークンが取り消されたときに、特定のエラーコードを送信するようになりました。 • 軽微なバグを修正。
iOS Broadcast SDK 1.18.0	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/1.18.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.18.0/ios • SDK は、接続されたステージが AWS コントロールプレーンによって削除されたとき、または使用中のトークンが取り消されたときに、特定のエラーコードを送信するようになりました。 • IVSCamera `setVideoZoomFactor` メソッドと関連する `IVSCameraDelegate` メソッドを追加しました。

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	5.275 MB	13.279 MB
armeabi-v7a	4.573 MB	9.254 MB
x86_64	5.472 MB	14.142 MB
x86	5.664 MB	14.554 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	3.393 MB	7.916 MB

2024 年 5 月 6 日

IVS Broadcast SDK: Web 1.11.0 (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 1.11.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • SDK がステージ `DISCONNECT` で復旧を試みなかったエッジケースを修正しました。 • `join()` タイムアウトエラーのエラーメッセージを更新しました。「10 秒後に InitialConnectTimedOut」の代わりに、SDK が「オペレーションがタイムアウトしました」を返すようになりました。

2024 年 4 月 30 日

IVS Broadcast SDK: Web 1.10.1 (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
Web Broadcast SDK 1.10.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 軽微なバグを修正。

2024 年 4 月 30 日

Amazon IVS Broadcast SDK: Android 1.15.2、iOS 1.15.2 (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
Android Broadcast SDK 1.15.2	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.15.2/android • 軽微なバグを修正。このバージョンへのアップグレードは、特定の理由がある場合にのみ行ってください。それ以外の場合は、リリースされた最新のバージョンを使用してください。
iOS Broadcast SDK 1.15.2	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/1.15.2/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.15.2/ios • 軽微なバグを修正。このバージョンへのアップグレードは、特定の理由がある場合にのみ

プラットフォーム	ダウンロードおよび変更
	行ってください。それ以外の場合は、リリースされた最新のバージョンを使用してください。

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	5.244 MB	13.198 MB
armeabi-v7a	4.543 MB	9.192 MB
x86_64	5.437 MB	14.051 MB
x86	5.631 MB	14.461 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	3.359 MB	7.836 MB

2024 年 4 月 22 日

Amazon IVS Broadcast SDK: Android 1.17.0、iOS 1.17.0 (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
Android Broadcast SDK 1.17.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.17.0/android

プラットフォーム	ダウンロードおよび変更
	• 配信中に発生する可能性のあるまれなクラッシュを修正しました。
iOS Broadcast SDK 1.17.0	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/1.17.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.17.0/ios • `AmazonIVSBroadcast` フレームワークに、Apple で義務付けられているプライバシーマニフェストが含まれるようになりました。

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	5.273 MB	13.275 MB
armeabi-v7a	4.571 MB	9.251 MB
x86_64	5.468 MB	14.137 MB
x86	5.662 MB	14.549 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	3.388 MB	7.916 MB

2024 年 3 月 21 日

Amazon IVS Broadcast SDK: Android 1.16.0、iOS 1.16.0、Web 1.10.0 (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 1.10.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • ステージのサブスクライブ解除または退出後に接続をクリーンアップする際の断続的なエラーを修正しました。
Android Broadcast SDK 1.16.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.16.0/android • Android 14 を搭載した Samsung デバイスの Exynos バリアントでプレビューがフリーズするのを修正しました。 • カメラズーム機能をクエリし、ズーム係数を設定する機能を追加しました。
iOS Broadcast SDK 1.16.0	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/1.16.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.16.0/ios • 軽微なバグを修正。

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	5.253 MB	13.21 MB
armeabi-v7a	4.551 MB	9.204 MB
x86_64	5.447 MB	14.070 MB
x86	5.640 MB	14.480 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	3.361 MB	7.836 MB

2024 年 3 月 13 日

Amazon IVS Broadcast SDK: Android 1.15.1、iOS 1.15.1 (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 1.15.1	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.15.1/android • リモート参加者をサブスクライブする際のまれなクラッシュを修正しました。
iOS Broadcast SDK 1.15.1	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/1.15.1/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/1.15.1/ios • リモート参加者をサブスクライブする際のまれなクラッシュを修正しました。

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	5.243 MB	13.194 MB
armeabi-v7a	4.541 MB	9.188 MB
x86_64	5.628 MB	14.455 MB
x86	5.434 MB	14.046 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	3.358 MB	7.820 MB

2024 年 3 月 13 日

サーバーサイドコンポジション API の更新

GridConfiguration に新しいプロパティと新しいピクチャインピクチャレイアウトを導入し、コンポジションのカスタマイズオプションを強化しました。特定のドキュメントの変更については、「ドキュメント履歴」(API リファレンスの変更の表) を参照してください 。

重要: アプリケーションが、タイルのサイズや位置など、現在のレイアウトの特定の機能に依存していないことを確認してください。レイアウトの視覚的な改善は、いつでも導入できます。

2024 年 3 月 8 日

サーバーサイドコンポジションのレイアウトの更新

本日、2024 年 2 月 7 日のエントリで説明されているデフォルトのグリッドレイアウトの変更を有効にしました。

2024 年 2 月 22 日

Amazon IVS Broadcast SDK: Android 1.15.0、iOS 1.15.0、Web 1.9.0 (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 1.9.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference • 内部エラー処理が改善されました。
Android Broadcast SDK 1.15.0	リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.15.0/android • 軽微なバグを修正。
iOS Broadcast SDK 1.15.0	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/1.15.0/AmazonIVSBroadcast-Stages.xcframework.zip リファレンスドキュメント: https://aws.github.io/amazon-ivs-broadcast-docs/1.15.0/ios • `IVSImagePreviewView` で新しいインスタンスを作成できるようにする拡張機能 `AVPictureInPictureController` を追加しました。 • `IVSImageDevice` に新しい API を追加し、デバイスがレンダリングする

プラットフォーム	ダウンロードおよび変更
	`AVSampleBufferDisplayLayer` を作成しました。 • iOS 17 以降を実行しているデバイスのビットレートが低い問題を修正しました。 • 軽微なバグを修正。

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	5.243 MB	13.194 MB
armeabi-v7a	4.541 MB	9.188 MB
x86_64	5.628 MB	14.455 MB
x86	5.434 MB	14.046 MB

Broadcast SDK サイズ: iOS

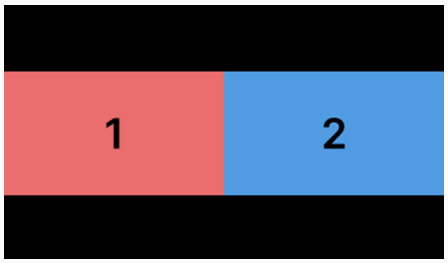
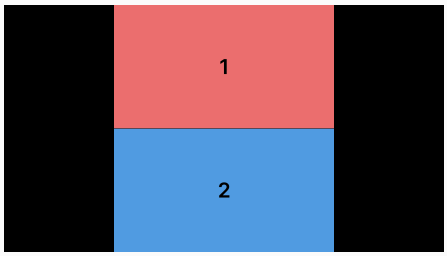
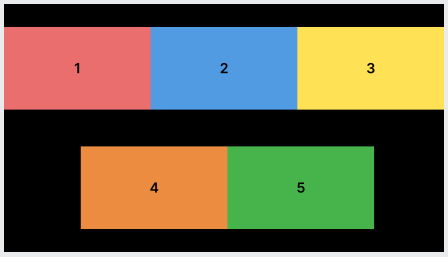
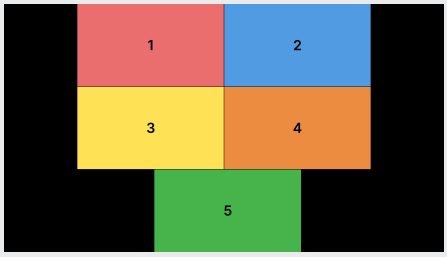
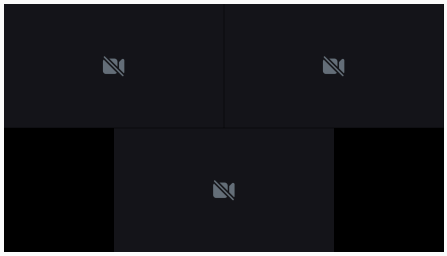
アーキテクチャ	圧縮サイズ	非圧縮サイズ
arm64	3.358 MB	7.820 MB

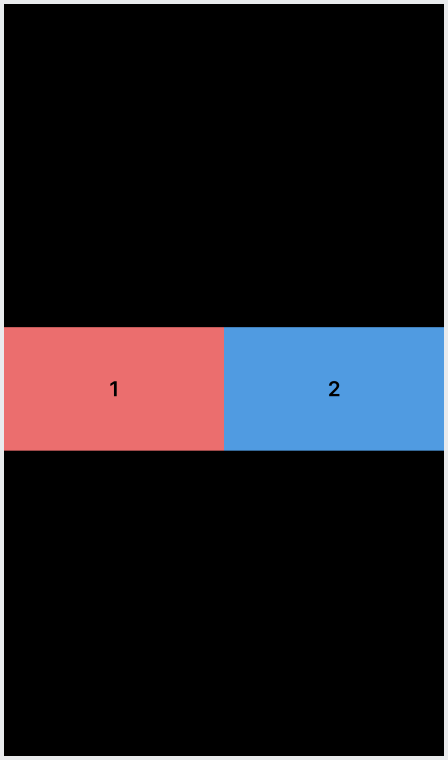
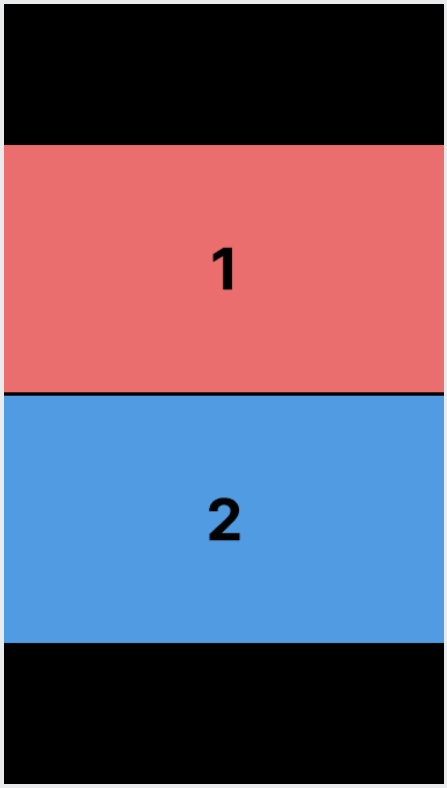
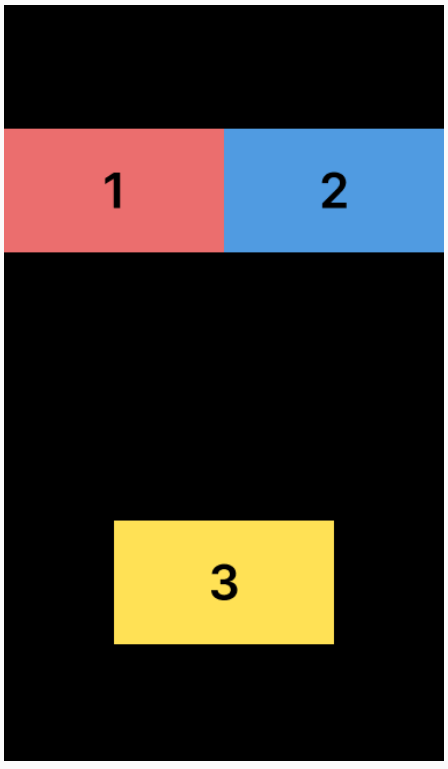
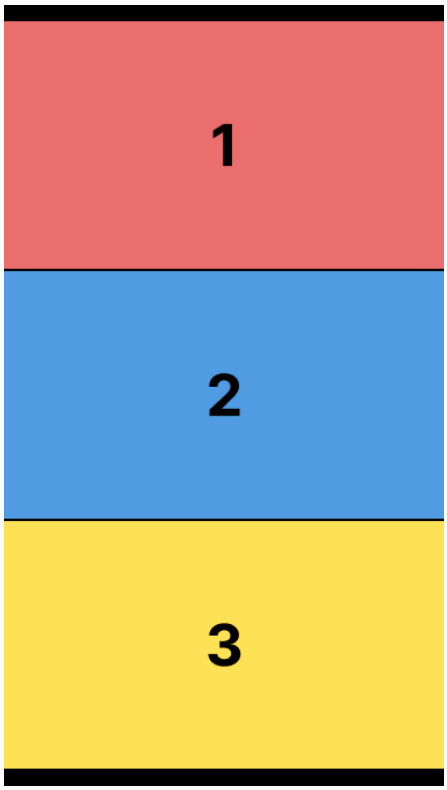
2024 年 2 月 7 日

サーバーサイドコンポジションのレイアウトの更新

このリリースでは、デフォルトのグリッドレイアウトに視覚的な改善が導入されました。これらの変更により、動画の表示方法が最適化され、空白スペースが削減されます。これらの変更は、2024 年 3 月 7 日に有効になります。

重要: アプリケーションが、タイルのサイズや位置など、現在のレイアウトの特定の機能に依存していないことを確認してください。レイアウトの視覚的な改善は、いつでも導入できます。

変更点の説明	Old	新
ビデオサイズを最大化するために、参加者の最適な配置を自動的に選択します。		
ギャップを減らし、黒い棒を最小限に抑えることで、スペースの使用率を向上させます。		
ビデオを共有していない参加者が明確に見えるように、新しい「カメラオフ」インジケータが追加されました。		

変更点の説明	Old	新
ポートレートで使用する場合のスペース使用率と割合を改善します。		
参加者間のスペースを最小限に抑え、レターボックスまたはピラーボックスを減らすことで、ポートレートで使用する場合のスペース使用率を向上させます。		

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 Real-Time Streaming にも配信できます。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」の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はデフォルトで無効になっています。 • 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、(ERRORED、次いで CONNECTING ではなく) DISCONNECTED の状態の STAGE_CONNECTION_STATE_CHANGED イベントを出力するようになりました。 • 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きに配信が失敗する問題を修正しました。
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

プラットフォーム	ダウンロードおよび変更
	• 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無効になっています。 • libWebRTC を M108 から M119 に更新しました。 • 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。 • ステレオ配信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。 • セッション参加後に参加者から黒い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、シンボルの競合を避けるために内部 libWebRTC 参照を更新しました。

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用のダウンロード: 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はデフォルトで無効になっています。 • libWebRTC を M108 から M119 に更新しました。 • 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。 • ステレオ配信のサポートを追加しました。これは、IVSLocalStageStreamAudioConfiguration を通じて有効にできます。 • 他の参加者に対して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

アーキテクチャ	圧縮サイズ	非圧縮サイズ
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の動画作成時間を短縮しました。 • `minAudioBitrateKbps` プロパティを削除しました (未使用でした)。 • インターネットの停止や変更時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、エコーキャンセレーション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。

プラットフォーム	ダウンロードおよび変更
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 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、安定性と回復性を重視して、録画と再生の両方のオーディオエンジンが改善されました。これにより、使用中のルート変更のサポートが強化され、エッジケースのバッテリー回復が改善され、メインスレッドのブロック量が削減されます。 • ステージからデタッチされて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問題を修正しました。(SDK は、その時点で受信オーディオ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) メトリックの名前をDownloadPacketLoss (Stage) に変更しました。IVS Real-Time Streaming 用の CloudWatch メトリクスも追加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配信を有効/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。

プラットフォーム	ダウンロードおよび変更
iOS Broadcast SDK 1.13.2	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/1.13.2/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 • 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

プラットフォーム	ダウンロードおよび変更
	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 または配信の解像度 (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用のダウンロード: 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 • 配信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 Real-Time Streaming 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 Real-Time Streaming 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、RecordingConfiguration、S3DestinationConfiguration、S3Detail、S3StorageConfigura を追加しました。また、3 つのオブジェクト (Composition、Destination、DestinationConfiguration) を変更しました。これは GetComposition レスポンスと StartComposition リクエストおよび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 Real-Time Streaming 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 Real-Time Streaming で Amazon EventBridge を使用する – 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 Composition エンドポイントと EncoderConfiguration エンドポイント、および 11 のオブジェクト (ChannelDestinationConfiguration、Composition、CompositionSummary、Destination、DestinationConfig を追加しました。

「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 Real-Time Streaming 用の CloudWatch メトリクスがリリースされました。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 Real-Time Streaming 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 • 配信状態が `ERRORED` 状態になった場合に `refreshStrategy` で再配信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 • TypeScript 5 の内部 Maybe 型に関するバグを修正しました。 • サイマルキャストサポートの検出機能が向上しました。 • 配信時に発生する `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)	• 配信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用のダウンロード: 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 Real-Time Streaming ユーザーガイド」および「IVS Real-Time Streaming API リファレンス」を参照してください。