Guide de l'utilisateur du streaming en temps réel

Amazon IVS

Amazon IVS: Guide de l'utilisateur du streaming en temps réel

Table of Contents

Qu'est-ce qu'Amazon IVS Real-Time Streaming ?

Amazon Interactive Video Service (IVS) Real-Time Streaming vous offre tout ce dont vous avez besoin pour ajouter du son et de la vidéo en temps réel à vos applications.

Force :

- Latence en temps réel : créez des applications pour les cas d'utilisation sensibles à la latence, afin d'aider vos spectateurs à rester connectés et engagés grâce à IVS Real-Time Streaming. Diffusez des flux en direct avec une latence pouvant être inférieure à 300 millisecondes entre l'hôte et le spectateur.
- Haute concurrence : exploitez le potentiel d'interactions à grande échelle grâce à IVS Real-Time Streaming. Accueillez jusqu'à 10 000 spectateurs et autorisez jusqu'à 12 hôtes à monter vers la scène virtuelle.
- Optimisé pour les mobiles : IVS Real-Time Streaming est optimisé pour les cas d'utilisation mobiles et s'adresse à un large éventail d'appareils et de capacités réseau. En intégrant les kits SDK de diffusion Amazon IVS pour Android et iOS, vos utilisateurs peuvent interagir en tant qu'hôtes ou spectateurs et profiter de flux en direct de haute qualité sur leurs appareils mobiles.

Cas d'utilisation :

- Spots invités : créez des applications qui permettent aux animateurs de promouvoir leurs invités « vers la scène », transformant ainsi les spectateurs en animateurs pour des interactions en temps réel.
- Mode Versus (VS) : créez des expériences de compétitions côte à côte et permettez aux spectateurs de regarder les hôtes s'affronter en temps réel.
- Salles audio : invitez les auditeurs à participer à la conversation en tant qu'invités et favorisez un engagement plus profond dans vos salles audio.
- Enchères vidéo en direct : transformez les enchères en événements vidéo interactifs et préservez leur enthousiasme et leur intégrité grâce à une latence en temps réel.

En plus de la documentation du produit ici, consultez https://ivs.rocks/, un site dédié pour parcourir de contenu publié (démos, exemples de code, articles de blog), estimer les coûts et découvrir Amazon IVS via des démonstrations en direct.

Solution mondiale, contrôle régional

Streaming et visionnage à l'échelle mondiale

Vous pouvez utiliser Amazon IVS pour diffuser en continu auprès d'utilisateurs du monde entier :

- Lorsque vous diffusez en continu, Amazon IVS ingère automatiquement la vidéo à un emplacement près de chez vous.
- Les spectateurs peuvent regarder vos flux en direct dans le monde entier.

Autrement dit, le « plan de données » fonctionne à l'échelle mondiale. Le plan de données fait référence au streaming, à l'ingestion et à la visualisation du contenu.

Contrôle régional

Tandis que le plan de données d'Amazon IVS est mondial, le « plan de contrôle » est quant à lui régional. Le plan de contrôle fait référence à la console Amazon IVS, à l'API et aux ressources (scènes).

En d'autres termes, on peut dire qu'Amazon IVS est un « service AWS régional ». Autrement dit, les ressources Amazon IVS de chaque région sont indépendantes des ressources similaires dans d'autres régions. Par exemple, une scène que vous créez dans une région est indépendante des scènes créées dans d'autres régions.

Lorsque vous utilisez des ressources (par exemple, au moment de créer une scène), vous devez spécifier la région de création. Par la suite, lorsque vous gérez ces ressources, vous devez le faire à partir de la région d'origine.

Si vous utilisez …	Vous devez spécifier la région …
Console Amazon IVS	Utilisation de la liste déroulante Select a Region (Sélectionner une région) en haut à droite de la barre de navigation
API Amazon IVS	Utilisation du point de terminaison de service approprié Consultez la Référence de l'API Amazon IVS Real-Time Streaming.

Si vous utilisez …	Vous devez spécifier la région …
	(Si vous accédez à l'API via un kit SDK, configurez le paramètre `region` du kit SDK. Nous vous invitons à consulter la section Tools to Build on AWS.)
CLI AWS	Deux options s'offrent à vous : • Ajouter `--region <aws-region>` à la commande de la CLI. • Ajoutez la région à votre fichier de configuration AWS local.

Notez que, quelle que soit la région de création d'une scène vous pouvez diffuser sur Amazon IVS depuis n'importe où et les spectateurs peuvent regarder le contenu où qu'ils soient.

Mise en route avec le streaming en temps réel IVS

Ce document explique les étapes nécessaires à l'intégration du streaming en temps réel Amazon IVS dans votre application.

Rubriques

Introduction

Prérequis

Avant d'utiliser le streaming en temps réel pour la première fois, exécutez les tâches suivantes. Pour obtenir des instructions, consultez la section Mise en route avec le streaming à faible latence IVS.

- Créer un compte AWS
- Configurer les utilisateurs root et administratifs

Autres références

- Référence du SDK de diffusion Web IVS
- Référence du SDK de diffusion Android IVS
- Référence du SDK de diffusion iOS IVS
- Référence de l'API de streaming en temps réel IVS

Terminologie du streaming en temps réel

Terme	Description	
Étape	Un espace virtuel où les participants peuvent échanger des vidéos en temps réel.	
Host (Hôte)	Un participant qui envoie une vidéo locale vers la scène.	
Lecteur	Un participant qui reçoit la vidéo des hôtes.	
Participant	Un utilisateur connecté à la scène en tant qu'hôte ou spectateur.	
Jeton de participant	Un jeton qui authentifie un participant lorsqu'il rejoint une scène.	
SDK de diffusion	Une bibliothèque cliente qui permet aux participants d'envoyer et de recevoir des vidéos.	

Présentation des étapes

1. the section called "Définir des autorisations IAM" : créez une politique AWS Identity and Access Management (IAM) qui donne aux utilisateurs un ensemble d'autorisations de base et attribuez cette politique aux utilisateurs.
2. Créer une scène : créez un espace virtuel où les participants peuvent échanger des fichiers audio et vidéo en temps réel.
3. Distribuer des jetons aux participants : envoyez des jetons aux participants pour qu'ils puissent rejoindre votre scène.

4. Intégrer le SDK de diffusion IVS : ajoutez le SDK de diffusion à votre application pour permettre aux participants d'envoyer et de recevoir des vidéos : the section called "Web", the section called "Android" et the section called "iOS".
5. Publier une vidéo et s'y abonner : envoyez votre vidéo vers la scène et recevez des vidéos d'autres animateurs : the section called "Web", the section called "Android" et the section called "iOS".

Définir des autorisations IAM

Ensuite, vous devez créer une politique AWS Identity and Access Management (IAM) qui donne aux utilisateurs un ensemble d'autorisations de base (par exemple, pour créer une scène Amazon IVS et créer des jetons de participants) et affecter cette politique aux utilisateurs. Vous pouvez soit attribuer les autorisations lors de la création d'un nouvel utilisateur, soit ajouter des autorisations à un utilisateur existant. Les deux procédures sont données ci-dessous.

Pour plus d'informations (par exemple, pour en savoir plus sur les utilisateurs et les politiques IAM, comment attacher une politique à un utilisateur et comment limiter les actions des utilisateurs sur Amazon IVS), consultez :

- Création d'un utilisateur IAM dans le Guide de l'utilisateur IAM
- Les informations dans Sécurité d'Amazon IVS concernent l'IAM et les « politiques gérées pour IVS ».
- Les informations IAM contenues dans Sécurité Amazon IVS

Vous pouvez utiliser une politique gérée par AWS existante pour Amazon IVS ou créer une politique qui personnalise les autorisations que vous souhaitez accorder à un ensemble d'utilisateurs, de groupes ou de rôles. Les deux approches sont décrites ci-dessous.

Utiliser une politique existante pour les autorisations IVS

Dans la plupart des cas, vous souhaiterez utiliser une politique gérée par AWS pour Amazon IVS. Elles sont décrites en détail dans la section Managed Policies for IVS de Sécurité IVS.

- Utilisez la politique gérée par AWS `IVSReadOnlyAccess` pour permettre à vos développeurs d'applications d'accéder à tous les points de terminaison des API IVS Get and List (pour le streaming à faible latence et en temps réel).

- Utilisez la politique gérée par AWS `IVSFullAccess` pour permettre à vos développeurs d'applications d'accéder à tous les points de terminaison des API IVS (pour le streaming à faible latence et en temps réel).

Facultatif : créer une politique personnalisée pour les autorisations Amazon IVS

Procédez comme suit :

1. Connectez-vous à la console de gestion AWS et ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau de navigation, sélectionnez Politicies (Politiques), puis Create policy (Créer une politique). La fenêtre Spécifier les autorisations s'ouvre.
3. Dans la fenêtre Spécifier les autorisations, choisissez l'onglet JSON et copiez/collez la politique IVS suivante dans la zone de texte de l'Éditeur de politique. (La politique n'inclut pas toutes les actions Amazon IVS. Vous pouvez ajouter/supprimer (Autoriser/Refuser) des autorisations d'accès aux points de terminaison selon vos besoins. Consultez IVS Real-Time Streaming API Reference pour en savoir plus sur les points de terminaison IVS.)

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

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms",
                "cloudwatch:GetMetricData",
                "s3:DeleteBucketPolicy",
                "s3:GetBucketLocation",
                "s3:GetBucketPolicy",
                "s3:PutBucketPolicy",
                "servicequotas:ListAWSDefaultServiceQuotas",
                "servicequotas:ListRequestedServiceQuotaChangeHistoryByQuota",
                "servicequotas:ListServiceQuotas",
                "servicequotas:ListServices",
                "servicequotas:ListTagsForResource"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Toujours dans la fenêtre Spécifier les autorisations, choisissez Suivant (faites défiler la fenêtre vers le bas pour voir ceci). Une fenêtre Vérifier et créer s'ouvre.
5. Dans la fenêtre Vérifier et créer, saisissez un Nom de politique et ajoutez éventuellement une Description. Notez le nom de la politique, car vous en aurez besoin lors de la création des utilisateurs (ci-dessous). En bas de la fenêtre, choisissez Create policy (Créer la politique).
6. Vous êtes renvoyé à la fenêtre de la console IAM, où vous devriez voir une bannière confirmant que votre nouvelle politique a été créée.

Créer un nouvel utilisateur et ajouter des autorisations

Clés d'accès utilisateur IAM

Les clés d'accès IAM sont constituées d'un ID de clé d'accès et d'une clé d'accès secrète. Elles permettent de signer les demandes de manière programmatique auprès des services AWS. Si vous n'avez pas de clés d'accès, vous pouvez en créer à l'aide de la console de gestion AWS. En guise de bonnes pratiques, ne créez pas de clés d'accès pour l'utilisateur root.

La seule fois où vous pouvez visualiser ou télécharger une clé d'accès secrète est lors de sa création. Vous ne pouvez pas la récupérer par la suite. Toutefois, pour créer des clés d'accès, vous devez disposer des autorisations permettant d'effectuer les opérations IAM nécessaires.

Conservez toujours les clés d'accès en lieu sûr. Ne les partagez jamais avec des tiers (même si une demande semble provenir d'Amazon). Pour de plus amples informations, veuillez consulter Gestion des clés d'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Procédure

Procédez comme suit :

1. Dans le panneau de navigation, choisissez Utilisateurs, puis Créer un utilisateur. Une fenêtre Spécifier les détails de l'utilisateur s'ouvre.
2. Dans la fenêtre Spécifier les détails de l'utilisateur :
   a. Sous Détails de l'utilisateur, saisissez le nouveau Nom d'utilisateur à créer.
   b. Cochez Fournir un accès utilisateur à l'AWS Management Console.
   c. Sous Mot de passe de la console, sélectionnez Mot de passe généré automatiquement.
   d. Cochez Les utilisateurs doivent créer un nouveau mot de passe lors de la prochaine connexion.
   e. Choisissez Suivant. Une fenêtre Définir les autorisations s'ouvre.
3. Pour Définir les autorisations, sélectionnez Attacher directement les politiques. Une fenêtre Politiques d'autorisations s'ouvre.
4. Dans le champ de recherche, entrez le nom d'une politique IVS (soit une politique gérée par AWS, soit une politique personnalisée que vous avez créée précédemment). Lorsqu'elle est trouvée, cochez la case pour sélectionner la politique.
5. Choisissez Suivant (en bas de la fenêtre). Une fenêtre Vérifier et créer s'ouvre.
6. Dans la fenêtre Vérifier et créer, vérifiez que toutes les informations utilisateur sont correctes, puis choisissez Créer un utilisateur (en bas de la fenêtre).
7. La fenêtre Récupérer le mot de passe s'ouvre et contient les Informations de connexion à la console. Enregistrez ces informations de manière sécurisée pour pouvoir vous y référer ultérieurement. Une fois que vous avez termin, choisissez Retourner à la liste des utilisateurs.

Ajouter des autorisations à un utilisateur existant

Procédez comme suit :

1. Connectez-vous à la console de gestion AWS et ouvrez la console IAM à l'adresse https://console.aws.amazon.com/iam/.
2. Dans le panneau de navigation, choisissezUsers (Utilisateurs), puis choisissez un nom d'utilisateur existant à mettre à jour. (Choisissez le nom en cliquant dessus ; ne cochez pas la case de sélection.)
3. Sur la page Récapitulatif, dans l'onglet Autorisations, choisissez Ajouter des autorisations. La fenêtre Ajouter des autorisations s'ouvre.
4. Sélectionnez Attach existing policies directly (Attacher directement les politiques existantes). Une fenêtre Politiques d'autorisations s'ouvre.
5. Dans le champ de recherche, entrez le nom d'une politique IVS (soit une politique gérée par AWS, soit une politique personnalisée que vous avez créée précédemment). Lorsque la politique est trouvée, cochez la case pour la sélectionner.
6. Choisissez Suivant (en bas de la fenêtre). Une fenêtre Vérifier s'ouvre.
7. Dans la fenêtre Vérifier, sélectionnez Ajouter des autorisations (en bas de la fenêtre).
8. Sur la page Summary (Récapitulatif), confirmez que la politique IVS a été ajoutée.

Créer une étape

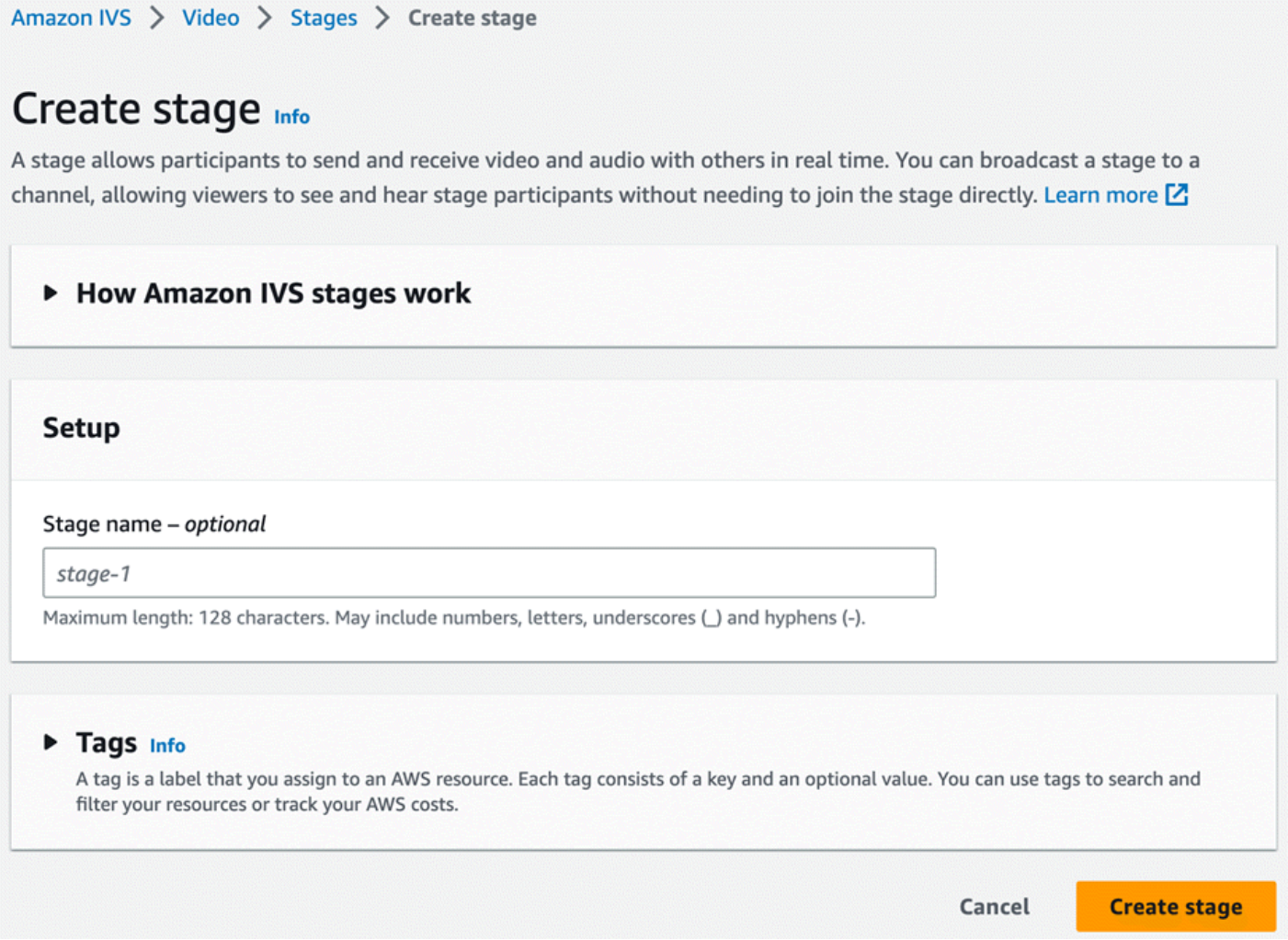
Une scène est un espace virtuel où les participants peuvent échanger des vidéos en temps réel. Il s'agit de la ressource de base de l'API de streaming en temps réel. Vous pouvez créer une étape à l'aide de la console ou du CreateStage point de terminaison.

Nous vous recommandons, dans la mesure du possible, de créer une scène pour chaque session logique et de le supprimer une fois terminé, plutôt que de conserver les anciennes scènes en vue d'une éventuelle réutilisation. Si les ressources périmées (anciennes scènes, à ne pas réutiliser) ne sont pas nettoyées, vous atteindrez probablement la limite du nombre maximum de scènes plus rapidement.

Instructions de la console

1. Ouvrez la console Amazon IVS.

   (Vous pouvez également accéder à la console Amazon IVS via la console de gestion AWS.)
2. Dans le panneau de navigation de gauche, sélectionnez Étapes, puis sélectionnez Créer une étape. La fenêtre Créer une étape s'affiche.



3. Saisissez éventuellement un Nom d'étape. Sélectionnez Créer une étape pour créer l'étape. La page de détails de l'étape s'affiche pour la nouvelle étape.

Instructions de la CLI

Pour installer l'AWS CLI, consultez la section Install or update the latest version of the AWS CLI.

Vous pouvez désormais utiliser l'interface de ligne de commande pour créer et gérer des ressources. L'API d'étape se trouve dans l'espace de noms ivs-realtime. Par exemple, pour créer une étape :

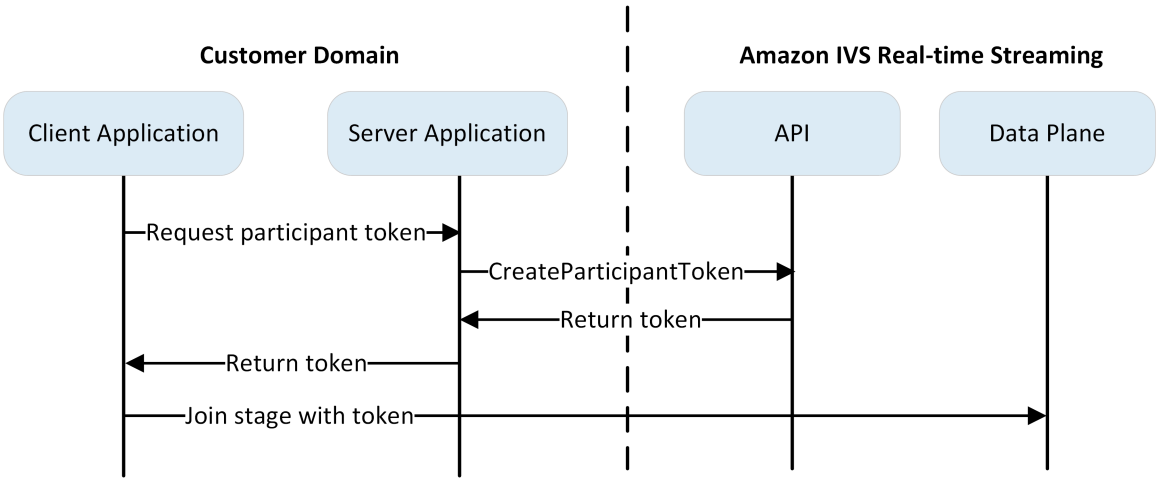
```
aws ivs-realtime create-stage --name "test-stage"
```

La réponse est :

```
{
```

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

Distribuer des jetons de participants



Maintenant que vous avez une scène, vous devez créer et distribuer des jetons aux participants pour leur permettre de rejoindre la scène et de commencer à envoyer et à recevoir des vidéos.

Comme indiqué ci-dessus, une application cliente demande un jeton à votre application serveur, et l'application serveur appelle à l' CreateParticipantToken aide d'un AWS SDK ou d'une demande signée SigV4. Étant donné que les informations d'identification AWS sont utilisées pour appeler l'API, le jeton doit être généré dans une application sécurisée côté serveur, et non dans l'application côté client.

Lors de la création d'un jeton de participant, vous pouvez éventuellement spécifier les fonctionnalités activées par ce jeton. La valeur par défaut est `PUBLISH` et `SUBSCRIBE`, qui permet au participant d'envoyer et de recevoir des fichiers audio et vidéo, mais vous pouvez émettre des jetons dotés d'un sous-ensemble de fonctionnalités. Par exemple, vous pouvez émettre un jeton n'ayant que la capacité `SUBSCRIBE` pour les modérateurs. Dans ce cas, les modérateurs peuvent voir les participants qui envoient des vidéos, mais ne peuvent pas envoyer leurs propres vidéos.

Vous pouvez créer des jetons participants via la console ou la CLI à des fins de test et de développement, mais il est fort probable que vous souhaitiez les créer à l'aide du SDK AWS dans votre environnement de production.

Vous aurez besoin d'un moyen de distribuer des jetons depuis votre serveur à chaque client (via une requête d'API par exemple). Nous ne proposons pas cette fonctionnalité. Pour ce guide, vous pouvez simplement copier et coller les jetons dans le code client en suivant les étapes suivantes.

Important : traitez les jetons comme des éléments opaques, c'est-à-dire ne créez pas de fonctionnalités basées sur le contenu des jetons. Le format des jetons pourrait changer à l'avenir.

Instructions de la console

1. Accédez à la scène que vous avez créée à l'étape précédente.
2. Sélectionnez Créer un jeton de participant. La fenêtre Créer un jeton de participant s'affiche.
3. Saisissez un ID utilisateur à associer au jeton. Il peut s'agir de n'importe quel texte codé en UTF-8.
4. Sélectionnez Créer un jeton de participant.
5. Copiez le jeton. Important : veillez à enregistrer le jeton. IVS ne le stocke pas et vous ne pourrez pas le récupérer ultérieurement.

Instructions de la CLI

La création d'un jeton avec l'AWS CLI nécessite que vous téléchargiez et configuriez d'abord la CLI sur votre machine. Pour plus de détails, consultez le Guide de l'utilisateur de l'Interface de ligne de commande AWS. Notez que la génération de jetons avec l'AWS CLI est utile à des fins de test, mais pour une utilisation en production, nous vous recommandons de générer des jetons côté serveur avec le SDK AWS (voir les instructions ci-dessus).

1. Exécutez la commande `create-participant-token` avec l'ARN de la scène. Incluez l'une ou l'ensemble des fonctions suivantes : `"PUBLISH"` ou `"SUBSCRIBE"`.

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

2. Cela renvoie un jeton de participant :

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

```
        "participantId": "tU06DT5jCJeb",
        "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtdmVFTm9sUyIsInJ
TaKjllW9Qac6c5xBrdAk"    }
}
```

3. Enregistrez ce jeton. Vous en aurez besoin pour rejoindre la scène et pour envoyer et recevoir des vidéos.

Instructions du SDK AWS

Vous pouvez utiliser le SDK AWS pour créer des jetons. Vous trouverez ci-dessous les instructions d'utilisation JavaScript du AWS SDK.

Important : ce code doit être exécuté côté serveur et sa sortie doit être transmise au client.

Prérequis : pour utiliser l'exemple de code ci-dessous, vous devez installer le package client-ivs-realtime aws-sdk/. Pour plus de détails, consultez Getting started with the AWS SDK for JavaScript.

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

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

Intégrer le SDK de diffusion IVS

IVS fournit un SDK de diffusion pour le Web, Android et iOS que vous pouvez intégrer à votre application. Le SDK de diffusion est utilisé à la fois pour envoyer et recevoir des vidéos. Dans cette section, nous écrivons une application simple qui permet à deux participants ou plus d'interagir en temps réel. Les étapes ci-dessous vous guident dans la création d'une application appelée BasicRealTime. Le code complet de l'application est activé CodePen et GitHub :

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

- iOS : https://github.com/aws-samples/ amazon-ivs-real-time - streaming-ios-samples

Web

Fichiers de configuration

Pour commencer, configurez vos fichiers en créant un dossier et un fichier HTML et JS initial :

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

Vous pouvez installer le SDK de diffusion à l'aide d'une balise de script ou de npm. Notre exemple utilise la balise script pour des raisons de simplicité, mais est facile à modifier si vous choisissez d'utiliser npm ultérieurement.

Utilisation d'une balise de script

Le SDK de diffusion Web est distribué sous forme de JavaScript bibliothèque et peut être consulté à l'adresse https://web-broadcast.live-video.net/1.8.0/ amazon-ivs-web-broadcast .js.

Lorsqu'elle est chargée via une balise `<script>`, la bibliothèque expose une variable globale dans la portée de la fenêtre nommée `IVSBroadcastClient`.

Utilisation de npm

Pour installer le package npm :

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

Vous pouvez désormais accéder à l'BroadcastClient objet IVS :

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

Android

Création du projet Android

1. Dans Android Studio, créez un New Project.

2. Choisissez Empty Views Activity.

   Remarque : dans certaines anciennes versions d'Android Studio, l'activité basée sur les vues est appelée Empty Activity. Si votre fenêtre Android Studio affiche Empty Activity et ne montre pas Empty Views Activity, sélectionnez Empty Activity. Sinon, ne sélectionnez pas Empty Activity, car nous utiliserons les API View (et non Jetpack Compose).

3. Donnez un Nom à votre projet, puis sélectionnez Finish.

Installation du SDK Broadcast

Pour ajouter la bibliothèque de diffusion Android Amazon IVS à votre environnement de développement Android, ajoutez la bibliothèque au fichier `build.gradle` de votre module, comme indiqué ici (pour la dernière version du SDK de diffusion Amazon IVS). Dans les nouveaux projets, le référentiel `mavenCentral` est peut-être déjà inclus dans votre fichier `settings.gradle`, si c'est le cas, vous pouvez omettre le bloc `repositories`. Pour notre exemple, nous devrons également activer la liaison de données dans le bloc `android`.

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
}

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

Vous pouvez également installer le kit SDK manuellement, en téléchargeant la dernière version à partir du lien suivant :

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

iOS

Création du projet iOS

1. Crée un projet Xcode.

2. Pour Platform, sélectionnez iOS.
3. Pour Application, sélectionnez App.
4. Saisissez le Product Name de votre application, puis sélectionnez Next.
5. Choisissez (naviguez jusqu'à) un répertoire dans lequel enregistrer le projet, puis sélectionnez Create.

Ensuite, vous devez importer le SDK. Nous vous recommandons d'intégrer le SDK de diffusion via CocoaPods. Vous pouvez également ajouter manuellement le cadre à votre projet. Les deux méthodes sont décrites ci-dessous.

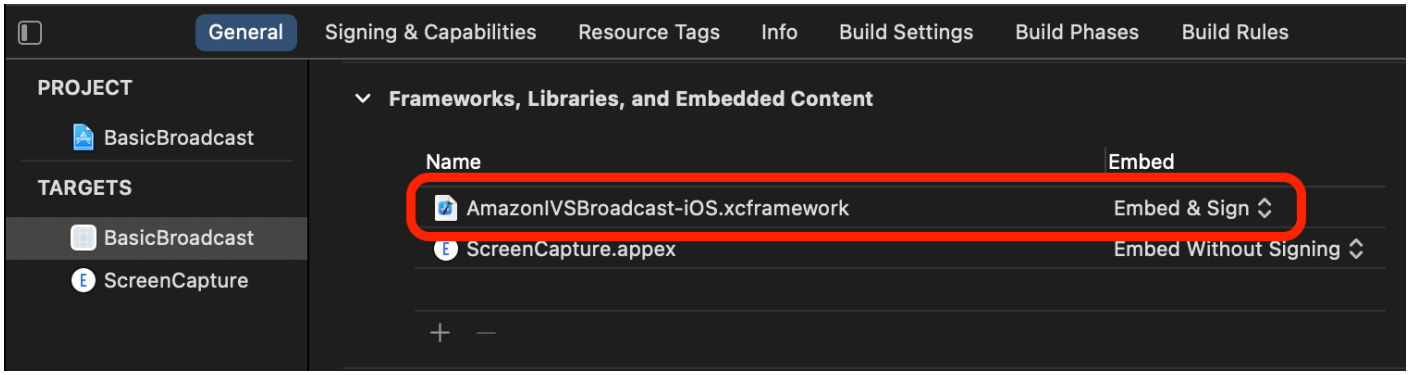
Recommandé : installez le SDK de diffusion () CocoaPods

En supposant que le nom de votre projet est `BasicRealTime`, créez un `Podfile` dans le dossier du projet avec le contenu suivant, puis exécutez `pod install` :

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

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

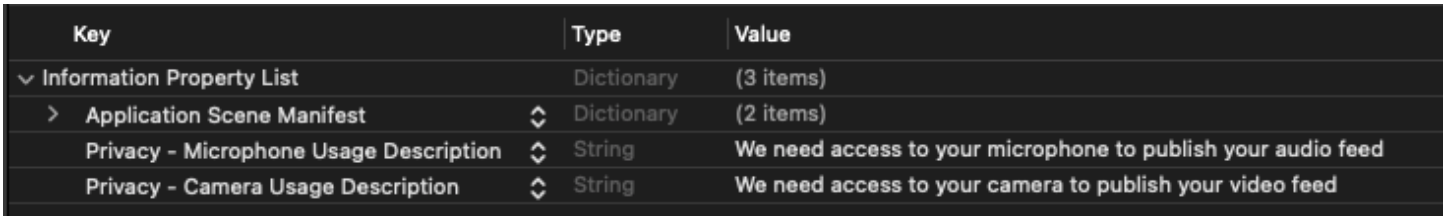
Autre approche : installer manuellement le cadre

1. Téléchargez la dernière version sur https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Extrayez le contenu de l'archive. `AmazonIVSBroadcast.xcframework` contient le kit SDK pour l'appareil et le simulateur.
3. Intégrez `AmazonIVSBroadcast.xcframework` en le faisant glisser dans la section Frameworks, Libraries, and Embedded Content (Cadre, bibliothèques et contenu intégré) de l'onglet General (Général) de votre cible d'application :



Configuration des autorisations

Vous devez mettre à jour votre projet `Info.plist` pour ajouter deux nouvelles entrées pour `NSCameraUsageDescription` et `NSMicrophoneUsageDescription`. Pour les valeurs, expliquez à l'utilisateur pourquoi votre application demande l'accès à la caméra et au microphone.



Publier une vidéo et s'y abonner

Consultez les informations ci-dessous pour le Web, Android et iOS.

Web

Créer un modèle de code HTML

Commençons par créer le modèle de code HTML et importons la bibliothèque sous forme de balise de script :

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

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

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

<body>

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

</body>
</html>
```

Accepter la saisie du jeton et ajouter des boutons Rejoindre/Quitter

Ici, nous remplissons le corps avec nos contrôles d'entrée. Ceux-ci prennent comme entrée le jeton, et ils configurent les boutons Rejoindre et Quitter. Généralement, les applications demandent le jeton à partir de l'API de votre application, mais dans cet exemple, vous allez copier et coller le jeton dans l'entrée du jeton.

```
<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 />
```

Ajouter des éléments de conteneur multimédia

Ces éléments hébergeront les médias pour nos participants locaux et distants. Nous ajoutons une balise de script pour charger la logique de notre application définie dans `app.js`.

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

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

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

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

Ceci termine la page HTML et vous devriez le voir lors du chargement du fichier `index.html` dans un navigateur :

Crée le fichier app.js

Passons à la définition du contenu de notre fichier `app.js`. Commencez par importer toutes les propriétés requises depuis le répertoire global du SDK :

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

Créer les variables d’application

Définissez des variables pour contenir les références aux éléments HTML de nos boutons Rejoindre et Quitter et pour stocker l’état de l’application :

```
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;
```

Créer joinStage 1 : définition de la fonction et validation de la saisie

La fonction `joinStage` prend le jeton d'entrée, crée une connexion à la scène et commence à publier la vidéo et l'audio extraits de `getUserMedia`.

Pour commencer, nous définissons la fonction et validons l'état et la saisie du jeton. Nous développerons cette fonction dans les prochaines sections.

```
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
};
```

Créer joinStage 2 : publier le contenu multimédia

Voici les médias qui seront publiés vers la scène :

```
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]);
```

Créer joinStage 3 : définir la stratégie de la scène et créer la scène

Cette stratégie de scène est au cœur de la logique de décision utilisée par le SDK pour décider du contenu à publier et des participants auxquels s'abonner. Pour plus d'informations sur l'objectif de la fonction, consultez la section Stratégie.

Cette stratégie est simple. Après avoir rejoint la scène, publiez les flux que vous venez de récupérer et abonnez-vous au son et à la vidéo de chaque participant distant :

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

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

Créer joinStage 4 : gérer les événements de la scène et le rendu multimédia

Les scènes génèrent de nombreux événements. Nous devrons écouter les événements `STAGE_PARTICIPANT_STREAMS_ADDED` et `STAGE_PARTICIPANT_LEFT` pour afficher et supprimer du contenu multimédia vers et depuis la page. Un ensemble plus complet d'événements est répertorié dans Événements.

Notez que nous créons ici quatre fonctions d'assistance pour nous aider à gérer les éléments DOM nécessaires : `setupParticipant`, `teardownParticipant`, `createVideoEl` et `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;
```

```
}
```

Créer joinStage 5 : rejoindre la scène

Complétons notre fonction `joinStage` en rejoignant enfin la scène !

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

Créer leaveStage

Définissez la fonction `leaveStage` qui sera invoquée par le bouton « Quitter ».

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

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

Initialiser les gestionnaires d'entrées et d'événements

Nous allons ajouter une dernière fonction à notre fichier `app.js`. Cette fonction est invoquée dès le chargement de la page et établit des gestionnaires d'événements pour rejoindre et quitter la scène.

```
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
```

Exécutez l'application et fournissez un jeton

À ce stade, vous pouvez partager la page Web localement ou avec d'autres personnes, ouvrir la page, saisir un jeton de participant et rejoindre l'étape.

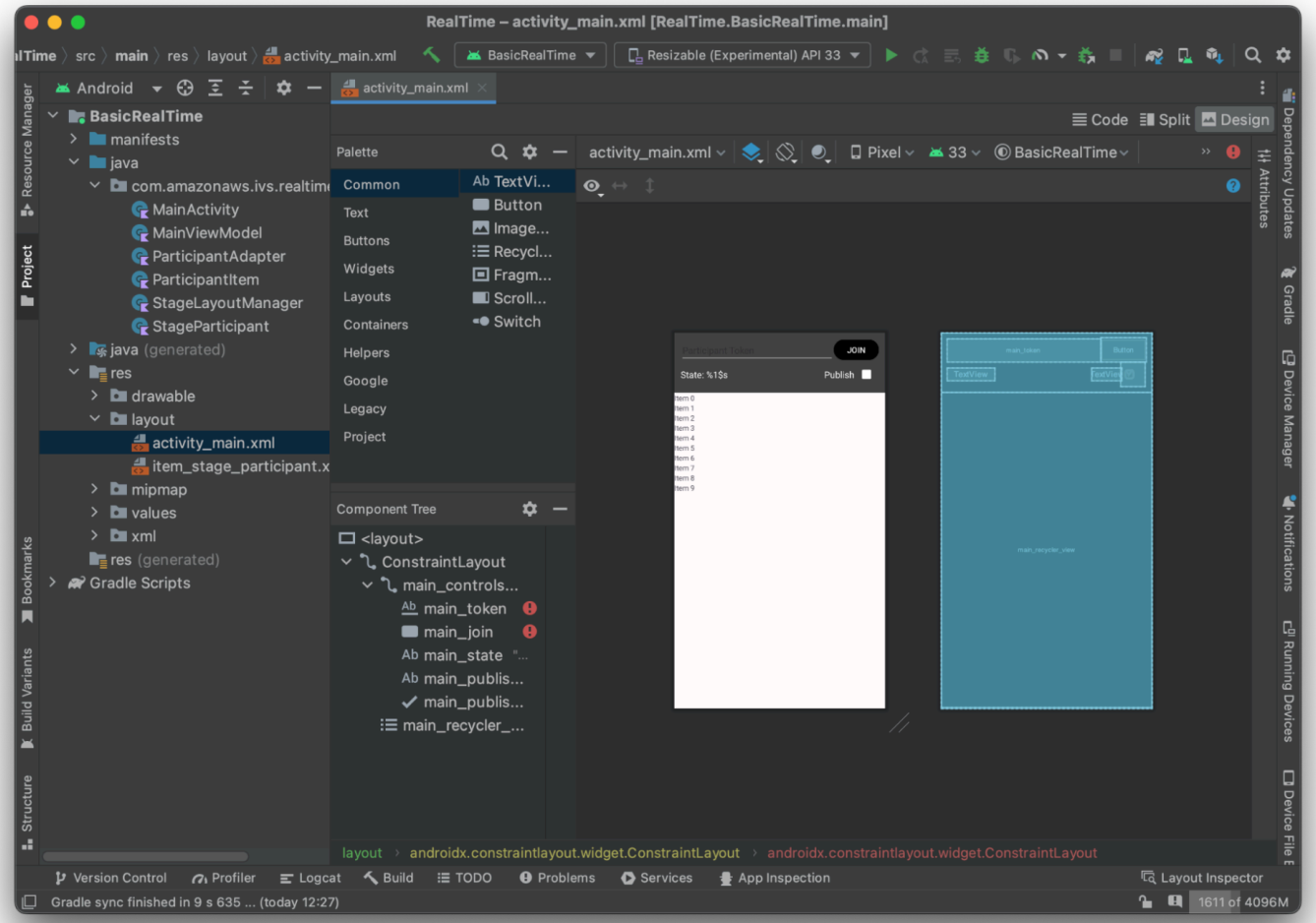
Quelle est la prochaine étape ?

Pour obtenir des exemples plus détaillés impliquant npm, React, etc., consultez le SDK de diffusion IVS : guide pour le Web (Streaming en temps réel).

Android

Créer les vues

Nous commençons par créer une disposition simple pour notre application à l'aide du fichier `activity_main.xml` créé automatiquement. La disposition contient un `EditText` pour ajouter un jeton, un `Button` Rejoindre, un `TextView` pour afficher l'état de la scène, et une `CheckBox` pour commuter l'état de publication.



Voici le code XML qui sous-tend la vue :

```
<?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>
```

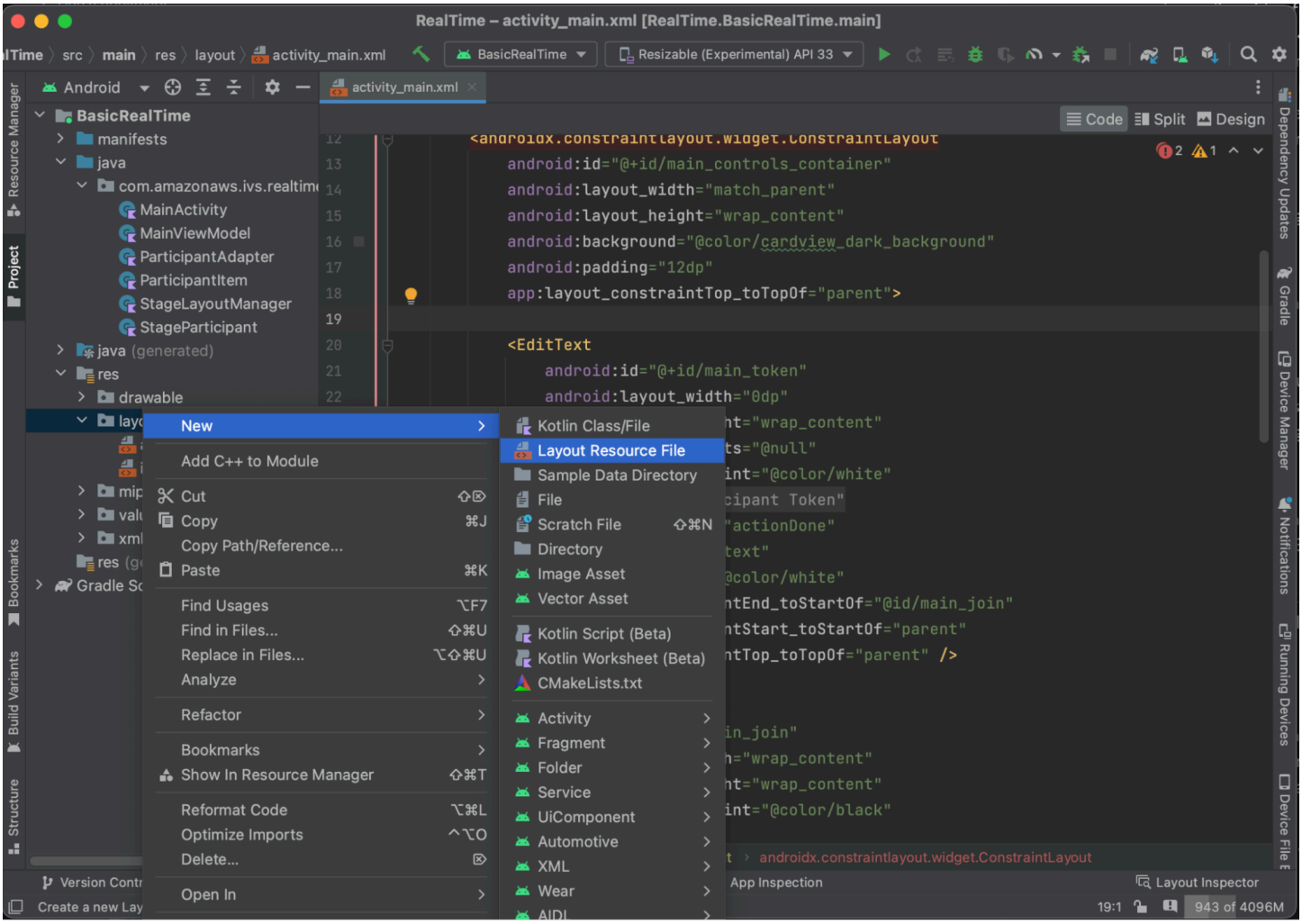
Nous avons référencé quelques ID de chaînes ici, nous allons donc créer notre fichier `strings.xml` entier maintenant :

```
<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>
```

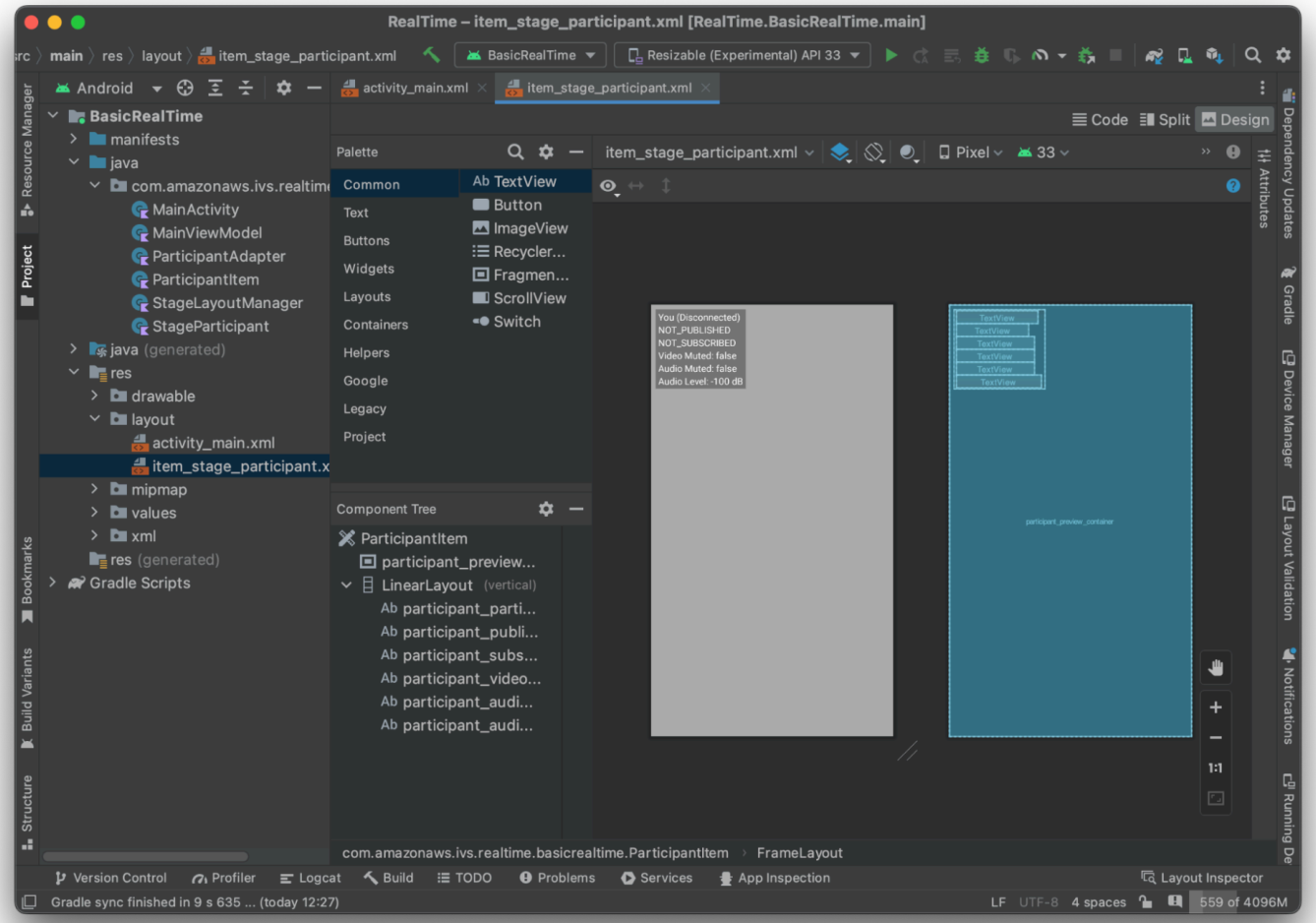
Lions ces vues dans le XML à notre `MainActivity.kt` :

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

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

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

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

Nous allons maintenant créer une vue d'élément pour notre `RecyclerView`. Pour ce faire, effectuez un clic droit sur votre répertoire `res/layout` et sélectionnez New > Layout Resource File. Nommez ce nouveau fichier `item_stage_participant.xml`.



La disposition de cet élément est simple : elle contient une vue permettant de rendre le flux vidéo d'un participant et une liste d'étiquettes permettant d'afficher des informations sur le participant :



Voici le XML :

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

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

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

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

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

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

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

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

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

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

    </LinearLayout>

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

Ce fichier XML renseigne une classe que nous n'avons pas encore créée : `ParticipantItem`. Étant donné que le XML inclut l'espace de nommage complet, veillez à mettre à jour ce fichier XML vers votre espace de nommage. Créons cette classe et configurons les vues, mais sinon, laissez-la vide pour le moment.

Créez une nouvelle classe Kotlin, `ParticipantItem` :

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

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

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

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

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

Autorisations

Pour utiliser la caméra et le microphone, vous devez demander des autorisations à l'utilisateur. Pour cela, nous suivons un flux d'autorisations standard :

```
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
    }
}
```

État de l’application

Notre application garde une trace des participants localement dans un `MainViewModel.kt` et l'état sera communiqué aux utilisateurs de `MainActivity` Kotlin. StateFlow

Créez une classe Kotlin, `MainViewModel` :

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

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

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

}
```

Dans `MainActivity.kt`, nous gérons notre modèle de vue :

```
import androidx.activity.viewModels

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

Pour utiliser `AndroidViewModel` et ces extensions Kotlin `ViewModel`, vous devrez ajouter ce qui suit au fichier `build.gradle` de votre module :

```
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 adaptateur

Nous allons créer une sous-classe `RecyclerView.Adapter` pour suivre nos participants et mettre à jour notre `RecyclerView` sur les événements de la scène. Mais d'abord, nous avons besoin d'une classe qui représente un participant. Créez une classe Kotlin, `StageParticipant` :

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

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

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

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

Nous utiliserons cette classe dans classe `ParticipantAdapter` que nous allons créer ensuite. Nous commençons par définir la classe et créer une variable pour suivre les participants :

```
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>()
```

Nous devons également définir notre `RecyclerView.ViewHolder` avant d'implémenter le reste des surcharges :

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

À partir de là, nous pouvons implémenter les surcharges `RecyclerView.Adapter` standard :

```
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)
```

```
    }
}
```

Enfin, nous ajoutons de nouvelles méthodes que nous appellerons à partir de notre `MainViewModel` lorsque des modifications seront apportées aux participants. Ces méthodes sont des opérations CRUD standard sur l'adaptateur.

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

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

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

En revenant au `MainViewModel`, nous devons créer et conserver une référence à cet adaptateur :

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

État de l'étape

Nous devons également suivre certains états de la scène au sein du `MainViewModel`. Définissons maintenant ces propriétés :

```
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>()
```

Pour voir votre propre aperçu avant de rejoindre une scène, nous créons immédiatement un participant local :

```
init {
    deviceDiscovery = DeviceDiscovery(application)

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

Nous voulons nous assurer de nettoyer ces ressources lorsque notre `ViewModel` est nettoyé. Nous surchargeons `onCleared()` immédiatement, afin de ne pas oublier de nettoyer ces ressources.

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

Maintenant, nous peuplons notre propriété de `streams` locaux dès que les autorisations sont accordées, en implémentant la méthode `permissionsGranted` que nous avons appelée plus tôt :

```
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)
    }
}
```

Implémentation du SDK Scène

Trois concepts de base sous-tendent la fonctionnalité temps réel : scène, stratégie et moteur de rendu. L'objectif de la conception consiste à minimiser la quantité de logique côté client nécessaire à la création d'un produit fonctionnel.

Stage.Strategy

L'implémentation de notre `Stage.Strategy` est simple :

```
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
}
```

Pour résumer, nous publions sur la base de notre état `publishEnabled`, et si nous publions, nous publierons les flux que nous avons collectés précédemment. Enfin, pour cet exemple, nous nous abonnons toujours aux autres participants, pour recevoir à la fois leur audio et leur vidéo.

StageRenderer

L'implémentation de `StageRenderer` est également assez simple, bien que compte tenu du nombre de fonctions, elle contienne un peu plus de code. L'approche générale de ce moteur de rendu consiste à mettre à jour notre `ParticipantAdapter` lorsque le SDK nous informe d'une modification apportée à un participant. Dans certains cas, nous traitons les participants locaux différemment, car nous avons décidé de les gérer nous-mêmes afin qu'ils puissent voir l'aperçu de leur caméra avant de rejoindre le groupe.

```
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) {}
}
```

Implémentation d'une personnalisation RecyclerView LayoutManager

La répartition des différents nombres de participants peut s'avérer complexe. Vous souhaitez qu'ils occupent l'intégralité du cadre de la vue parent, mais vous ne souhaitez pas gérer la configuration de chaque participant de manière indépendante. Pour vous faciliter la tâche, nous allons procéder à l'implémentation d'un `RecyclerView.LayoutManager`.

Créez une autre classe, `StageLayoutManager`, qui étend `GridLayoutManager`. Cette classe est conçue pour calculer la disposition de chaque participant en fonction du nombre de participants dans une disposition en ligne/colonne basée sur le flux. Chaque ligne a la même hauteur que les autres, mais les colonnes peuvent avoir des largeurs différentes par ligne. Voir le commentaire de code au-dessus de la variable `layouts` pour une description de la façon de personnaliser ce comportement.

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

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

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

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

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

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

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

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

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

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

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

De retour dans `MainActivity.kt`, nous devons définir l'adaptateur et le gestionnaire de disposition pour notre `RecyclerView` :

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

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

Connexion des actions de l’interface utilisateur

Nous nous rapprochons ; il ne nous reste plus que quelques actions d’interface utilisateur à connecter.

Tout d’abord, nous avons notre `MainActivity` qui observe les changements de `StateFlow` depuis le `MainViewModel` :

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

Ensuite, nous ajoutons des auditeurs à notre bouton Rejoindre et à la case à cocher Publier :

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

Les deux fonctionnalités d’appel ci-dessus sont disponibles dans notre `MainViewModel`, que nous implémentons maintenant :

```
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
}
```

Affichage des participants

Enfin, nous devons rendre les données que nous recevons du SDK sur l'élément participant que nous avons créé précédemment. La logique du `RecyclerView` est déjà terminée, il ne nous reste plus qu'à implémenter l'API `bind` dans `ParticipantItem`.

Nous allons commencer par ajouter une fonction vide, puis la parcourir étape par étape :

```
fun bind(participant: StageParticipant) {

}
```

Nous allons d'abord gérer l'état simplifié, l'identifiant du participant, l'état de publication et l'état d'abonnement. Pour ceux-ci, nous mettons simplement à jour nos `TextViews` directement :

```
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
```

Ensuite, nous mettrons à jour les états de coupure de l'audio et de la vidéo. Pour obtenir l'état de coupure, nous devons trouver `ImageDevice` et `AudioDevice` à partir du tableau de flux. Pour optimiser les performances, nous mémorisons les derniers identifiants des appareils connectés.

```
// 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"
}
```

Enfin, nous voulons afficher un aperçu du `imageDevice` :

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

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

Et nous affichons les statistiques audio du `audioDevice` :

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

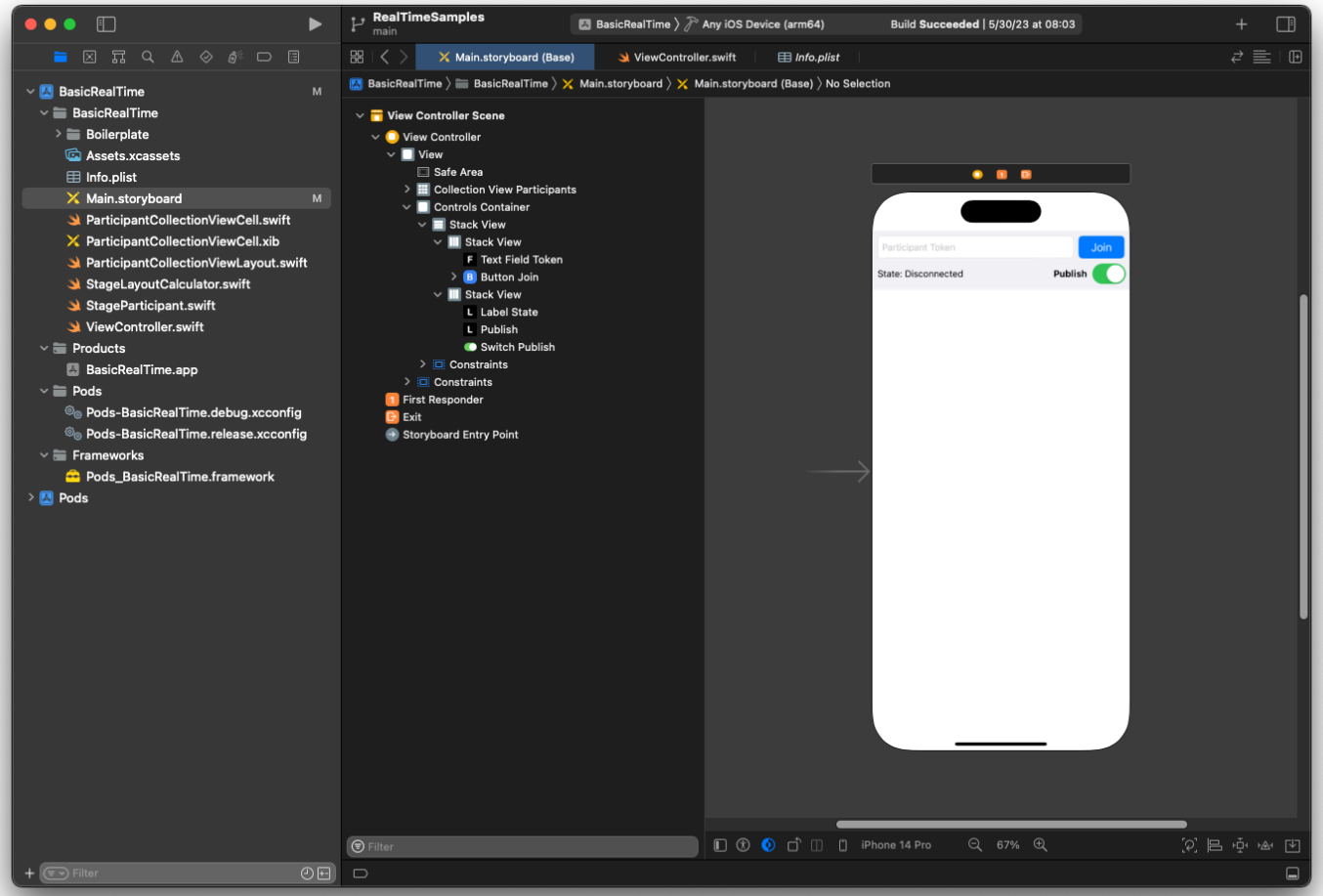
iOS

Créer les vues

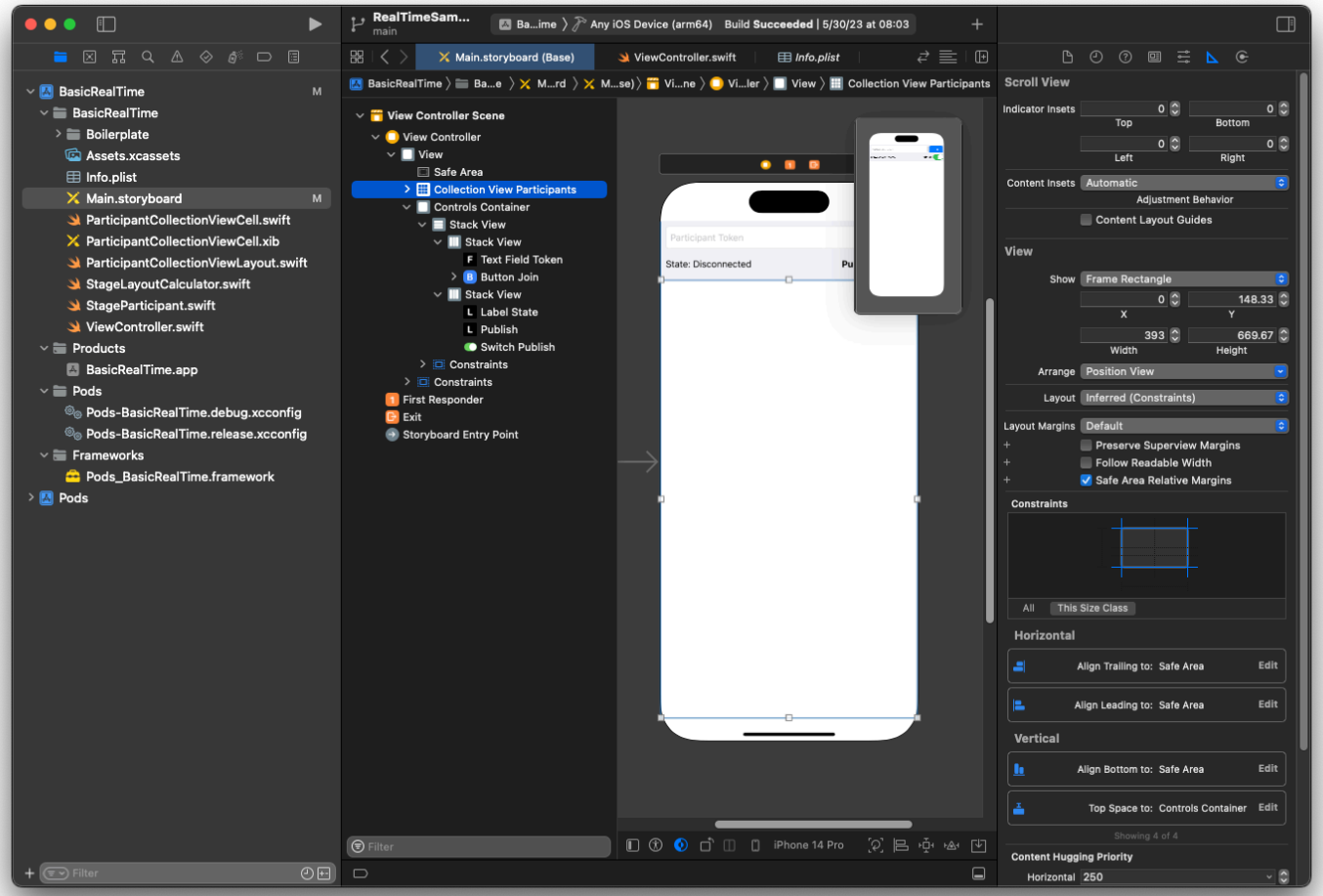
Nous commençons par utiliser le fichier `ViewController.swift` créé automatiquement pour importer `AmazonIVSBroadcast`, puis ajouter quelques `@IBOutlets` à lier :

```
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!
```

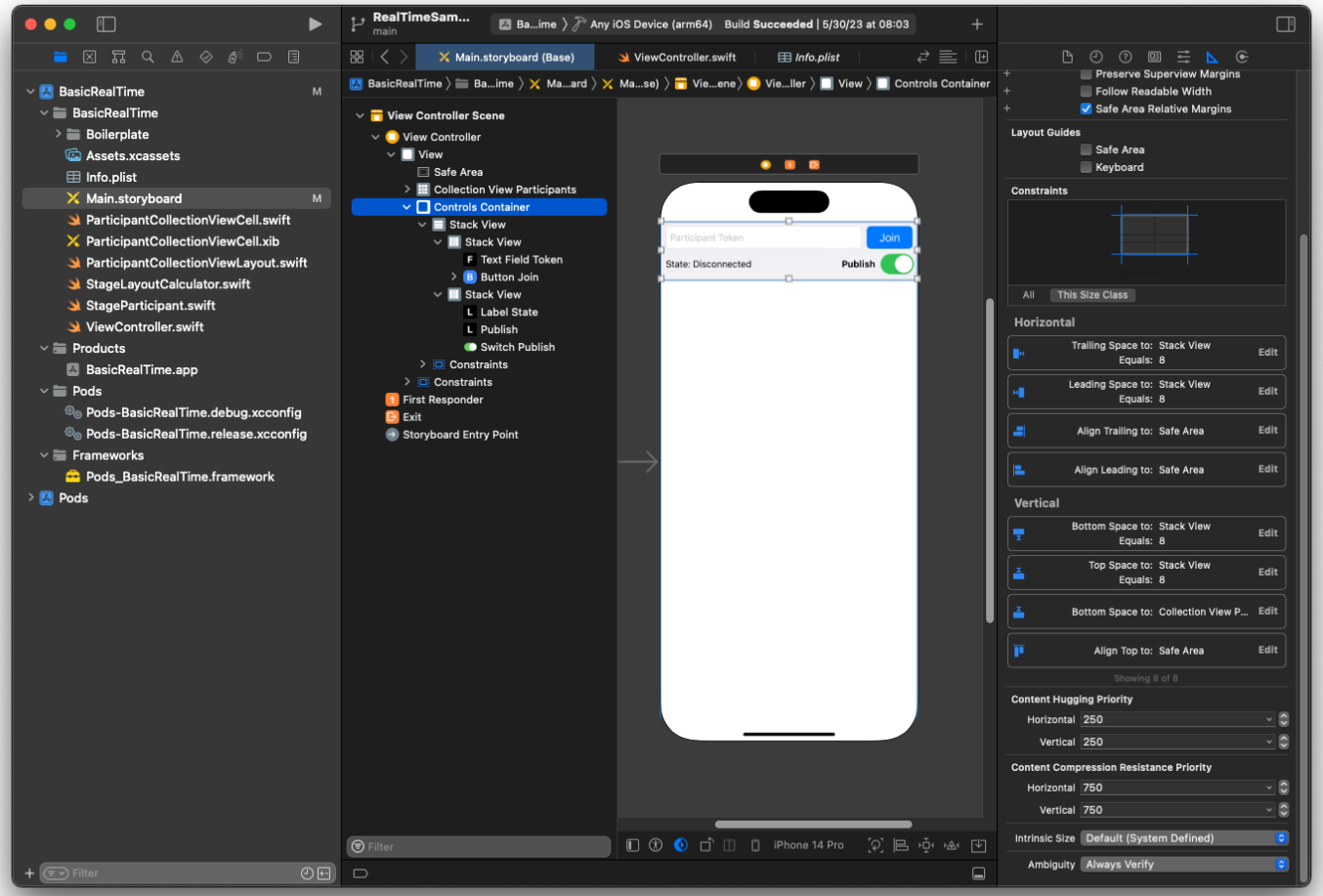
Nous créons maintenant ces vues et les relions dans `Main.storyboard`. Voici la structure de vue que nous allons utiliser :



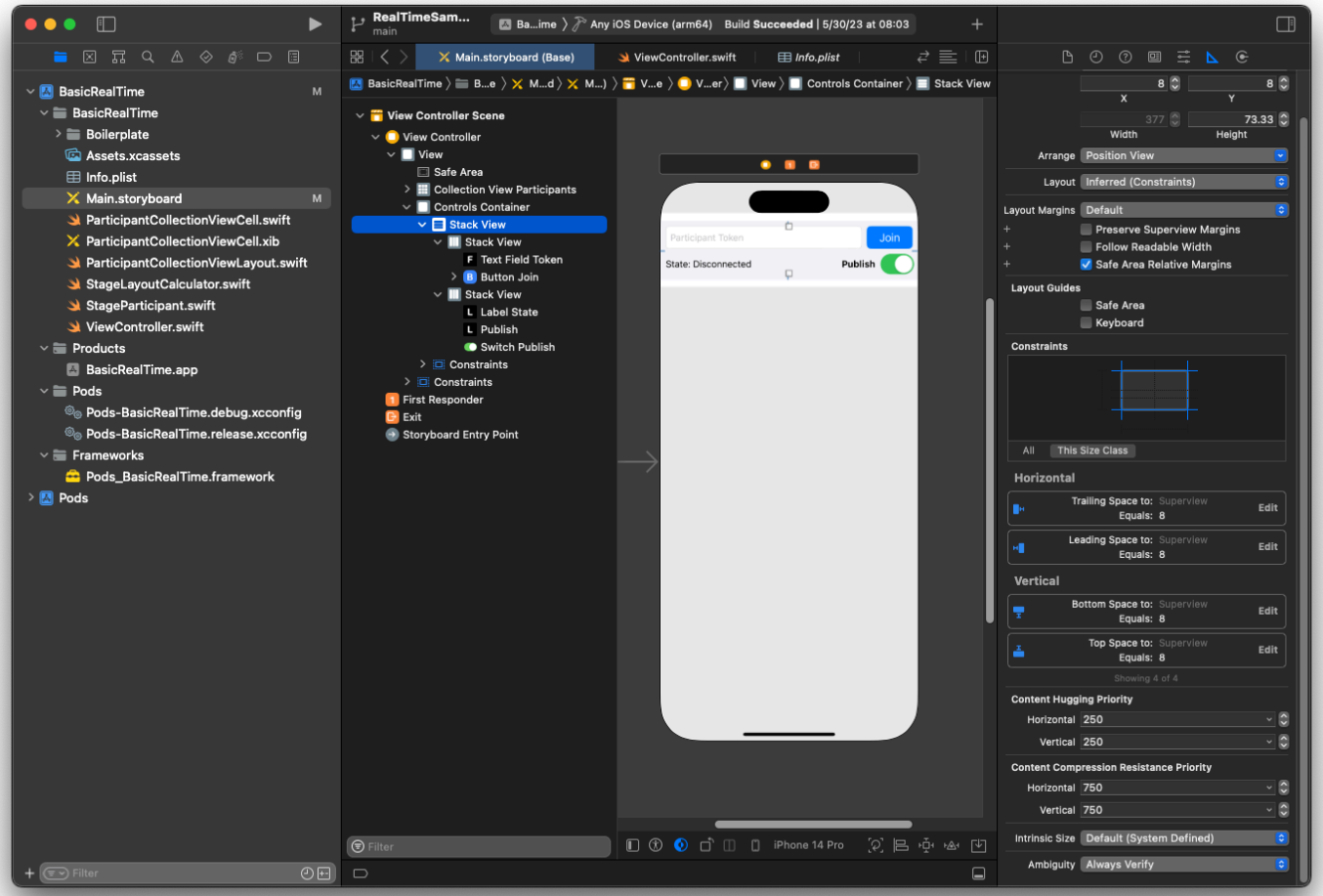
Pour AutoLayout la configuration, nous devons personnaliser trois vues. La première vue est Collection View Participants (une `UICollectionView`). Liez Leading, Trailing, et Bottom à Safe Area. Liez également Top à Controls Container.



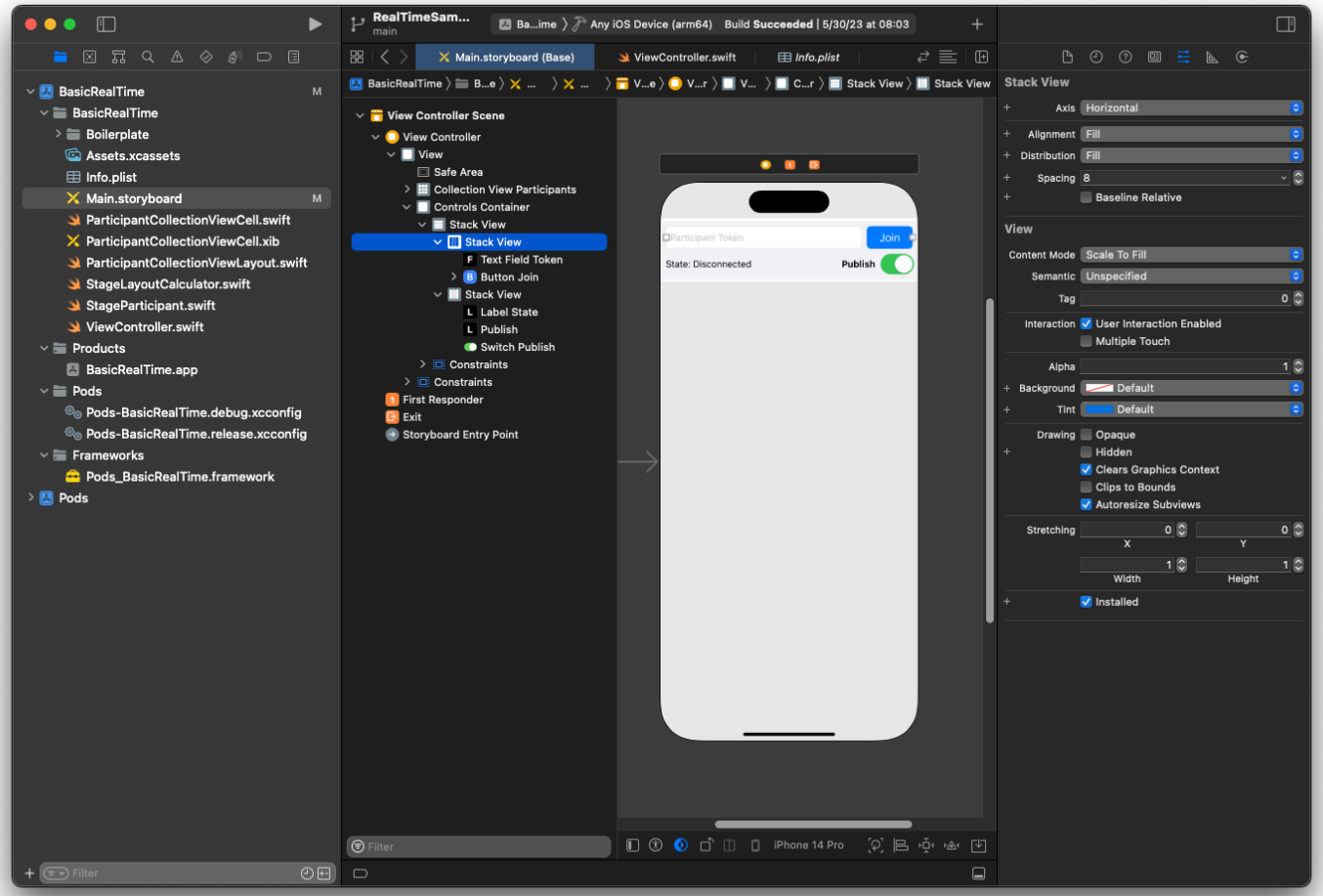
La deuxième vue est Controls Container. Liez Leading, Trailing, et Top à Safe Area :



La troisième et dernière vue est Vertical Stack View. Liez Top, Leading, Trailing, et Bottom à Superview. Pour le style, définissez l'espacement sur 8 au lieu de 0.



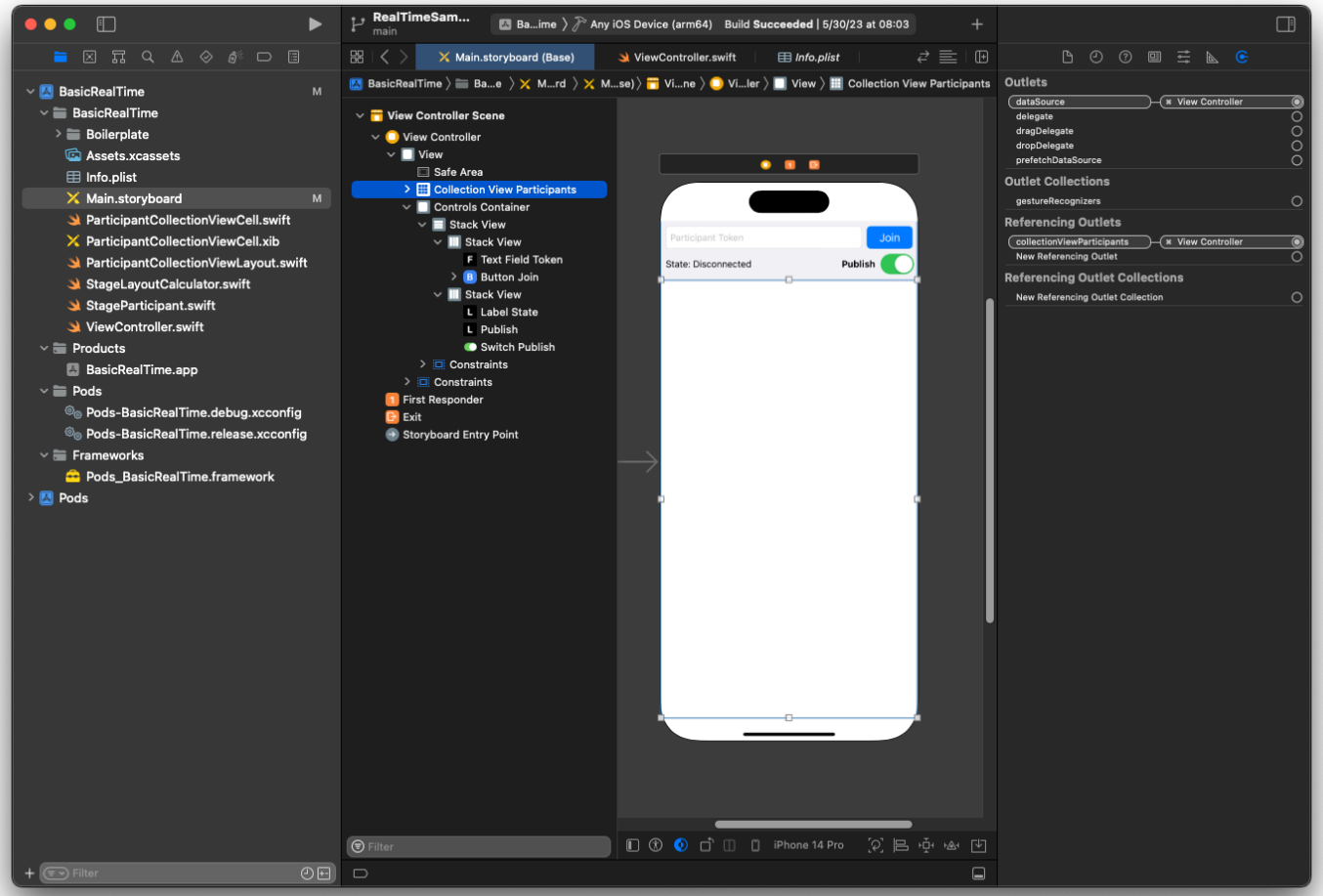
L'interface utilisateur StackViews gérera la mise en page des vues restantes. Pour les trois interfaces utilisateur StackViews, utilisez Fill comme alignement et distribution.



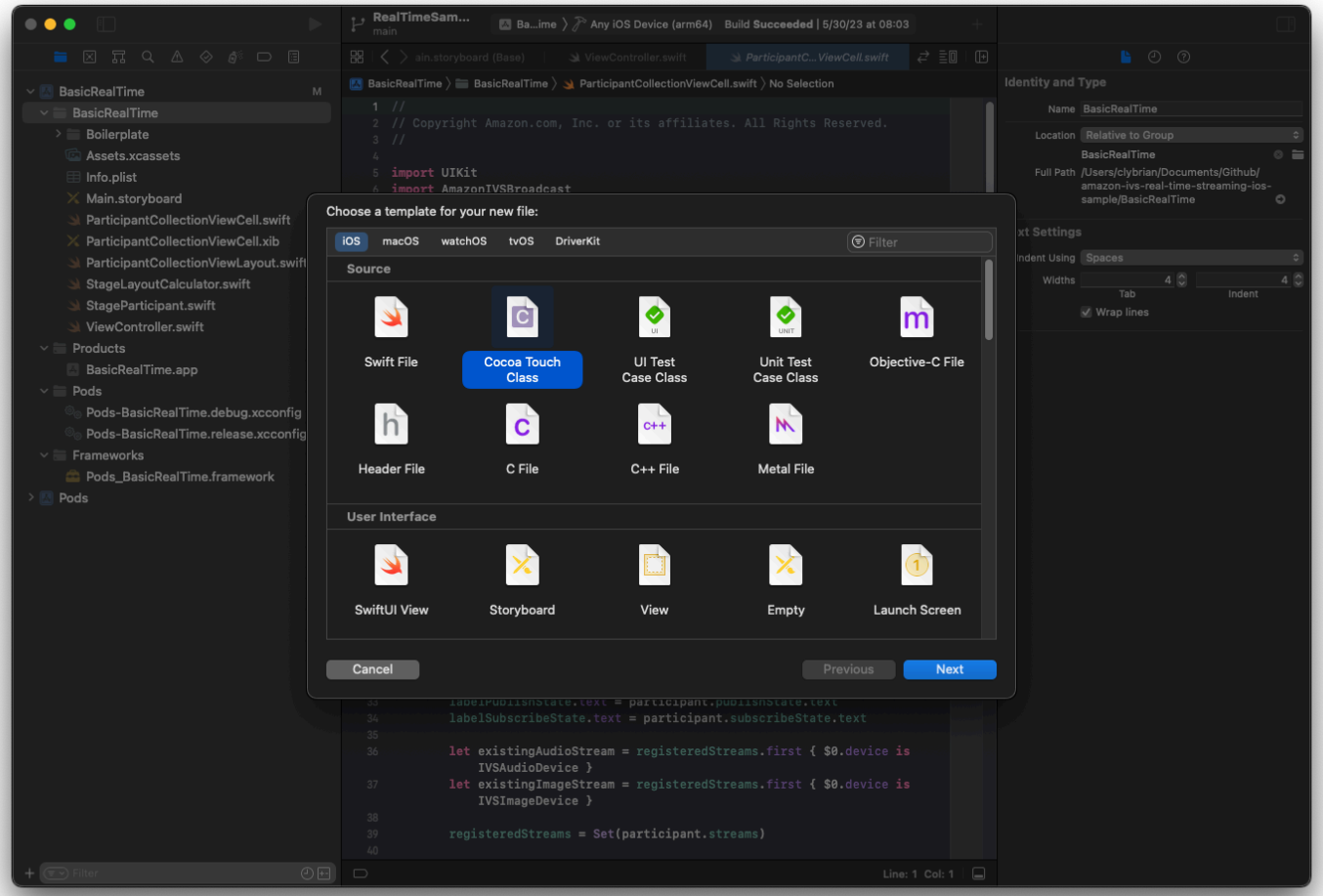
Enfin, relions ces vues à notre `ViewController`. De dessus, cartographiez les vues suivantes :

- Text Field Join est relié à `textFieldToken`.
- Button Join est relié à `buttonJoin`.
- Label State est relié à `labelState`.
- Switch Publish est relié à `switchPublish`.
- Collection View Participants est relié à `collectionViewParticipants`.

Profitez également de cette période pour définir la `dataSource` de l'élément Collection View Participants sur le `ViewController` propriétaire :



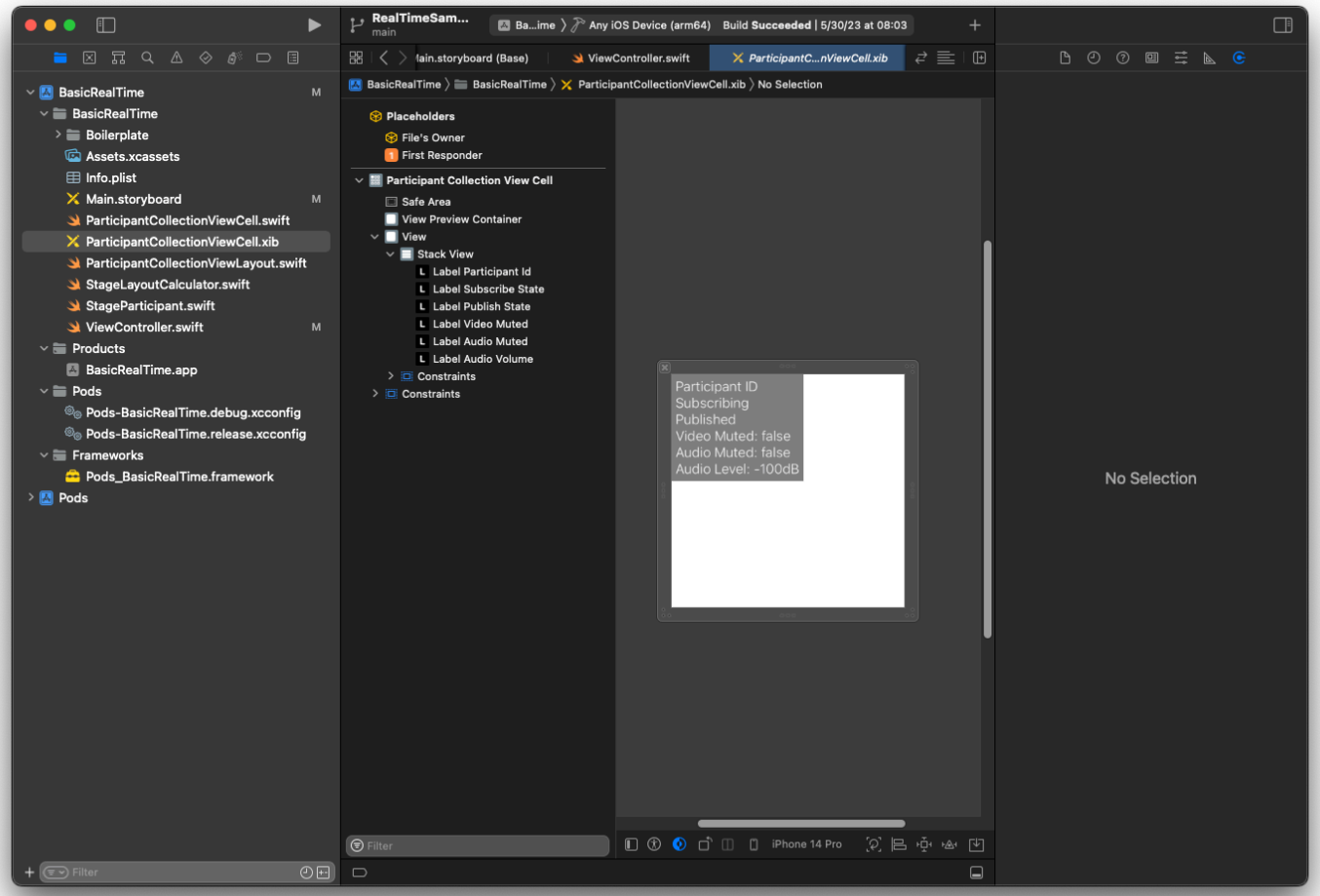
Nous créons maintenant la sous-classe `UICollectionViewCell` dans laquelle afficher les participants. Commencez par créer un fichier Cocoa Touch Class :



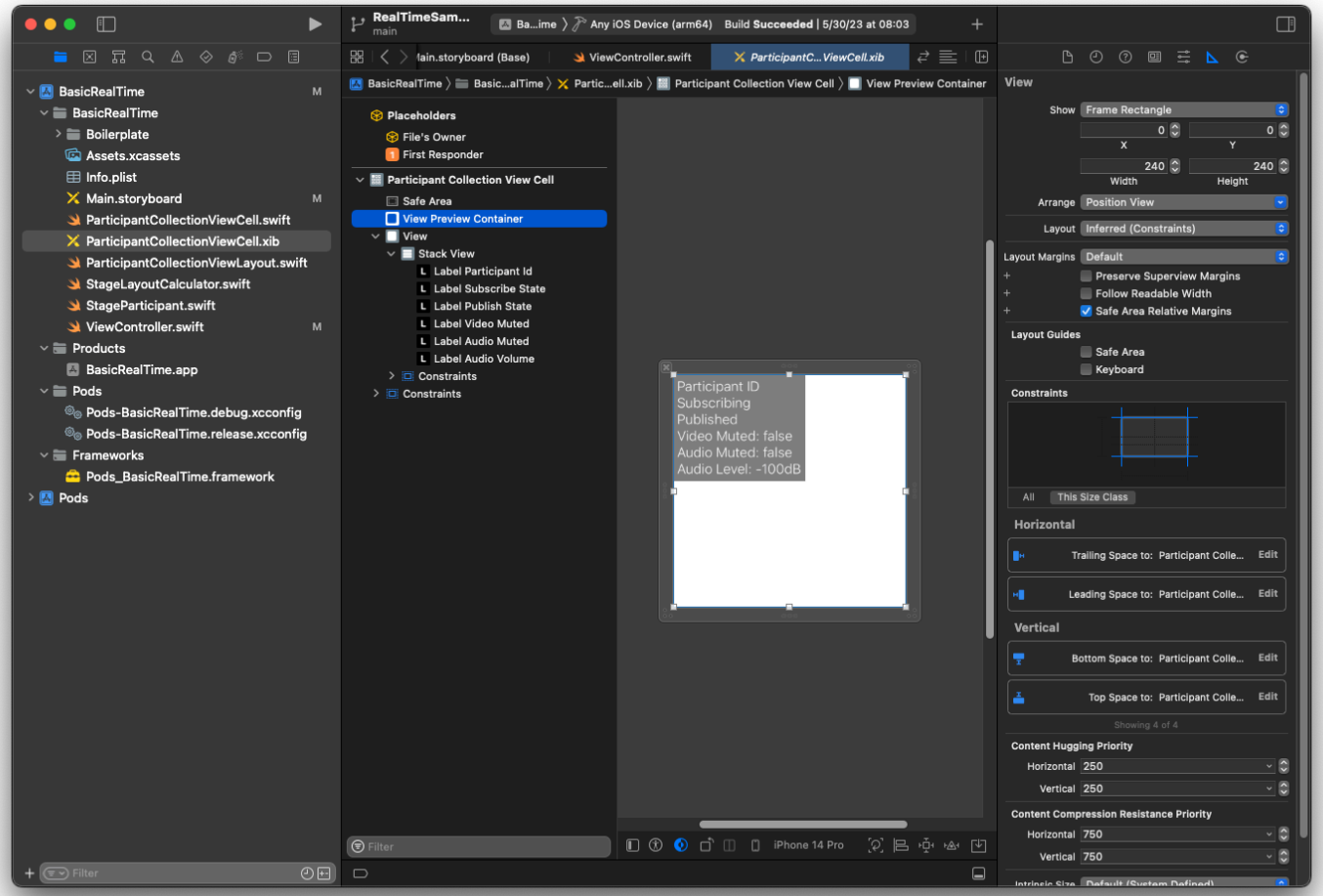
Nommez-le `ParticipantUICollectionViewCell` et faites-en une sous-classe de `UICollectionViewCell` dans Swift. Nous recommençons dans le fichier Swift, en créant nos `@IBOutlets` à lier :

```
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!
```

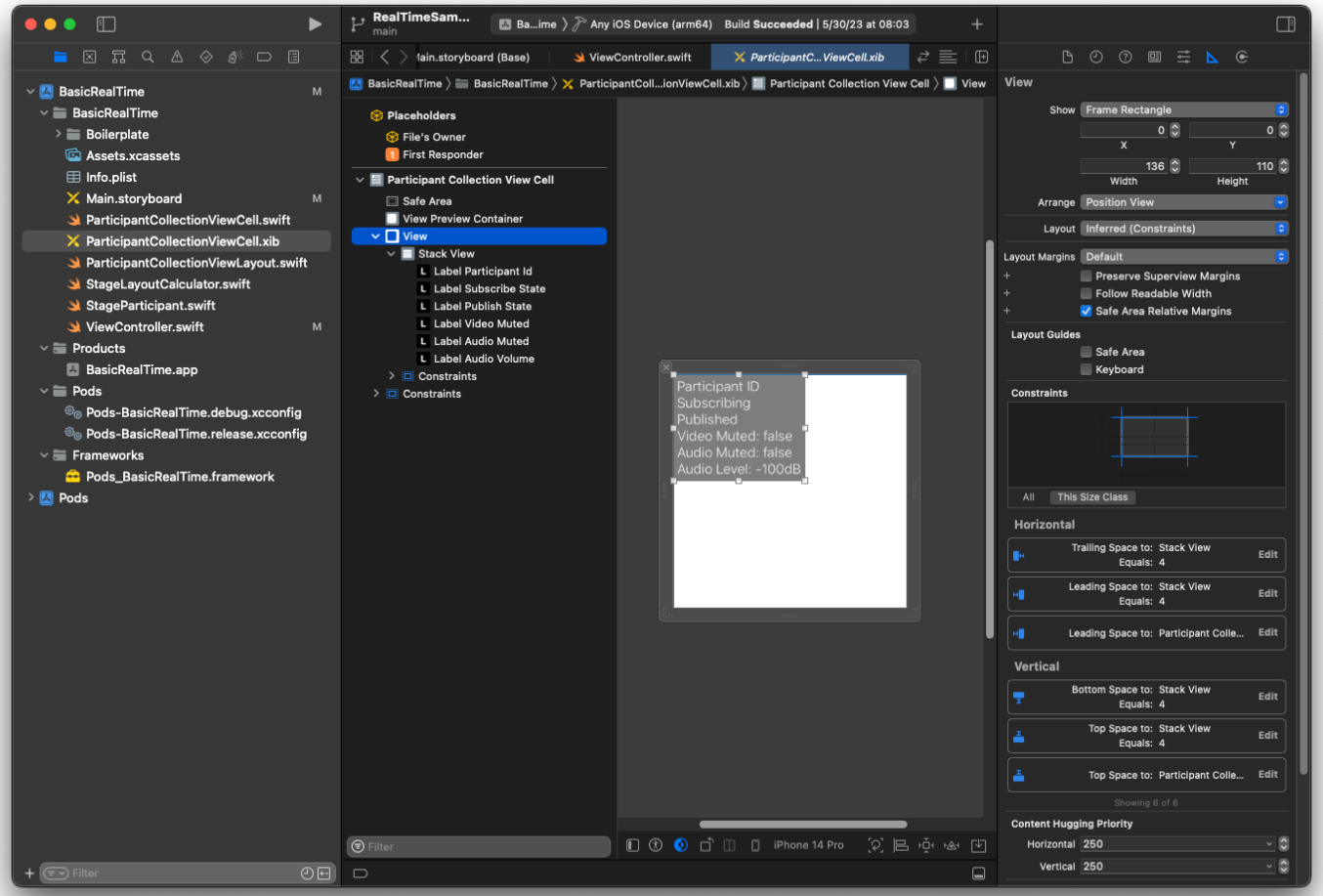
Dans le fichier XIB associé, créez cette hiérarchie de vues :



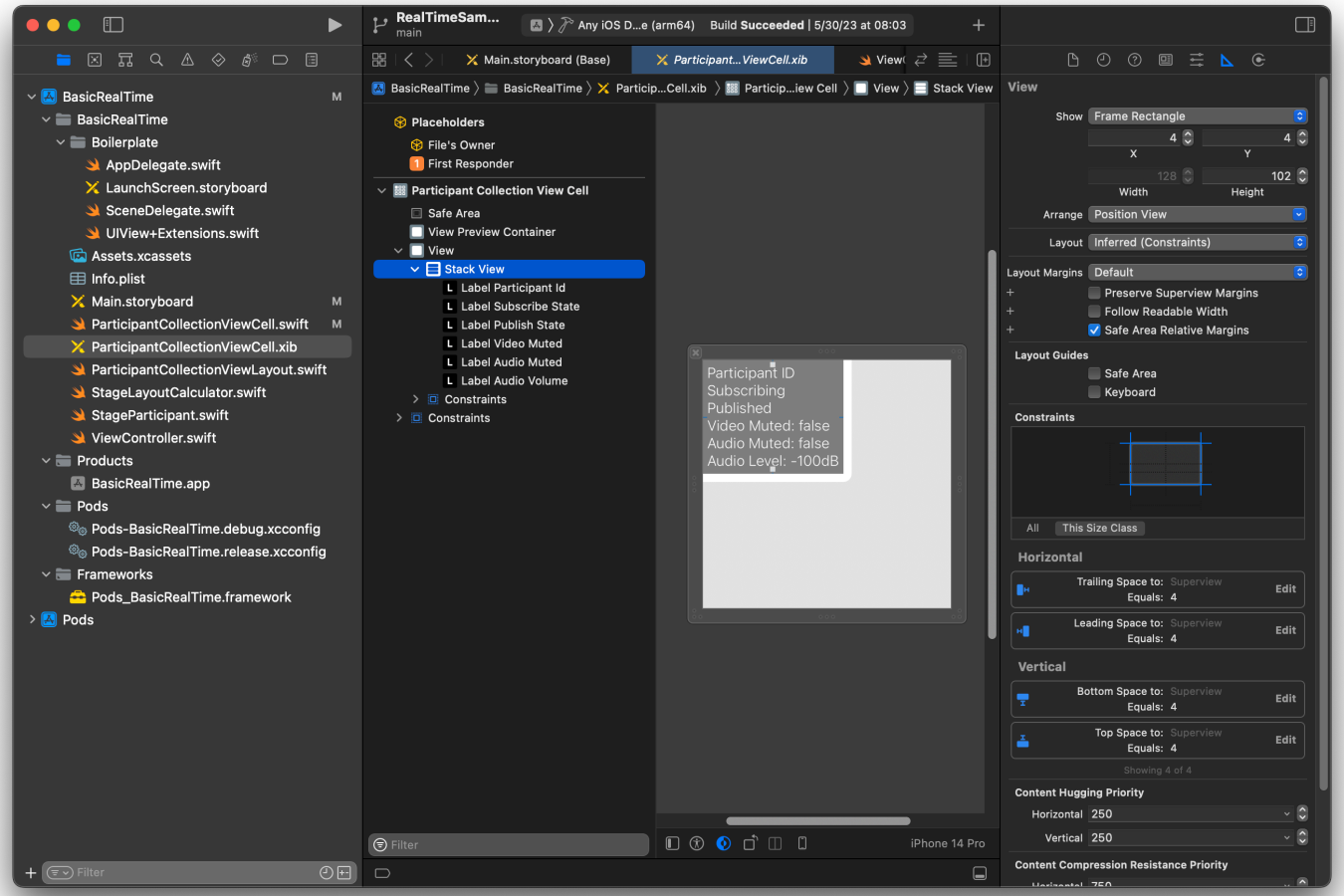
En effet AutoLayout, nous modifierons à nouveau trois vues. La première vue est View Preview Container. Liez Trailing, Leading, Top et Bottom à Participant Collection View Cell.



La deuxième vue est View. Liez Leading et Top à Participant Collection View Cell et définissez la valeur sur 4.



La troisième vue est Stack View. Liez Trailing, Leading, Top et Bottom à Superview et définissez la valeur sur 4.



Autorisations et minuteur d'inactivité

De retour à notre `ViewController`, nous allons désactiver le minuteur d'inactivité du système pour empêcher l'appareil de se mettre en veille pendant l'utilisation de notre application :

```
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
}
```

Ensuite, nous demandons les autorisations de caméra et de microphone auprès du système :

```
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)
    }
}
```

État de l’application

Nous devons configurer notre `collectionViewParticipants` avec le fichier de disposition que nous avons créé précédemment :

```
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")
}
```

Pour représenter chaque participant, nous créons une structure simple appelée `StageParticipant`. Cela peut être inclus dans le fichier `ViewController.swift`, ou un nouveau fichier peut être créé.

```
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
    }
}
```

Pour suivre ces participants, nous en conservons une liste en tant que propriété privée dans notre `ViewController` :

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

Cette propriété sera utilisée pour alimenter notre `UICollectionViewDataSource` qui était lié depuis le storyboard plus tôt :

```
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'")
        }
    }

}
```

Pour voir votre propre aperçu avant de rejoindre une scène, nous créons immédiatement un participant local :

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

Cela se traduit par le rendu d'une cellule de participant immédiatement après l'exécution de l'application, représentant le participant local.

Les utilisateurs veulent pouvoir se voir eux-mêmes avant de rejoindre une scène. Nous allons donc implémenter la méthode `setupLocalUser()` qui est appelée précédemment par le code de gestion des autorisations. Nous stockons la référence de la caméra et du microphone en tant qu'objets `IVSLocalStageStream`.

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

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

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

Ici, nous avons trouvé la caméra et le microphone de l'appareil via le SDK et les avons stockés dans notre objet local `streams`, puis nous avons assigné le tableau de `streams` du premier participant (le participant local que nous avons créé plus tôt) à notre `streams`. Enfin, nous appelons `participantsChanged` avec un `index` défini sur 0 et un `changeType` défini sur `updated`. Cette fonction est une fonction d'aide à la mise à jour de notre `UICollectionView` avec de belles animations. Voici à quoi cela ressemble :

```
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)])
    }
}
```

Ne vous préoccupez pas encore de `cell.set` ; nous y reviendrons plus tard, mais c'est là que nous afficherons le contenu de la cellule en fonction du participant.

Le `ChangeType` est une simple énumération :

```
enum ChangeType {
```

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

Enfin, nous voulons savoir si la scène est connectée. Nous utilisons un simple `bool` pour suivre cela, qui mettra automatiquement à jour notre interface utilisateur lorsqu'elle sera elle-même mise à jour.

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

Implémenter le SDK Scène

Trois concepts de base sous-tendent la fonctionnalité temps réel : scène, stratégie et moteur de rendu. L'objectif de la conception consiste à minimiser la quantité de logique côté client nécessaire à la création d'un produit fonctionnel.

IVS StageStrategy

L'implémentation de notre `IVSStageStrategy` est simple :

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

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

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

```
}
```

En résumé, nous ne publions que si l'interrupteur de publication est en position « activé », et si nous le faisons, seuls les flux que nous avons collectés précédemment sont publiés. Enfin, pour cet exemple, nous nous abonnons toujours aux autres participants, pour recevoir à la fois leur audio et leur vidéo.

IVS StageRenderer

L'implémentation de `IVSStageRenderer` est également assez simple, bien que compte tenu du nombre de fonctions, elle contienne un peu plus de code. L'approche générale de ce moteur de rendu consiste à mettre à jour notre tableau `participants` lorsque le SDK nous informe d'une modification apportée à un participant. Dans certains cas, nous traitons les participants locaux différemment, car nous avons décidé de les gérer nous-mêmes afin qu'ils puissent voir l'aperçu de leur caméra avant de rejoindre le groupe.

```
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)
    }
}
```

Ce code utilise une extension pour convertir l'état de connexion en texte convivial :

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

Implémentation d'une interface utilisateur personnalisée CollectionViewLayout

La répartition des différents nombres de participants peut s'avérer complexe. Vous souhaitez qu'ils occupent l'intégralité du cadre de la vue parent, mais vous ne souhaitez pas gérer la configuration de chaque participant de manière indépendante. Pour vous faciliter la tâche, nous allons procéder à l'implémentation d'un `UICollectionViewLayout`.

Créez un autre fichier, `ParticipantCollectionViewLayout.swift`, qui étend `UICollectionViewLayout`. Cette classe utilisera une autre classe appelée `StageLayoutCalculator`, que nous aborderons bientôt. La classe reçoit les valeurs de trame calculées pour chaque participant et génère ensuite les objets `UICollectionViewLayoutAttributes` nécessaires.

```
import Foundation
import UIKit

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

    private let layoutCalculator = StageLayoutCalculator()

    private var contentBounds = CGRect.zero
```

```
    private var cachedAttributes = [UICollectionViewLayoutAttributes]()

    override func prepare() {
        super.prepare()

        guard let collectionView = collectionView else { return }

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

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

    override var collectionViewContentSize: CGSize {
        return contentBounds.size
    }

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

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

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

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

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

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

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

        return attributesArray
    }

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

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

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

La classe `StageLayoutCalculator.swift` revêt une plus grande importance. Elle est conçue pour calculer les cadres de chaque participant en fonction du nombre de participants dans une disposition en ligne/colonne basée sur le flux. Chaque ligne a la même hauteur que les autres, mais

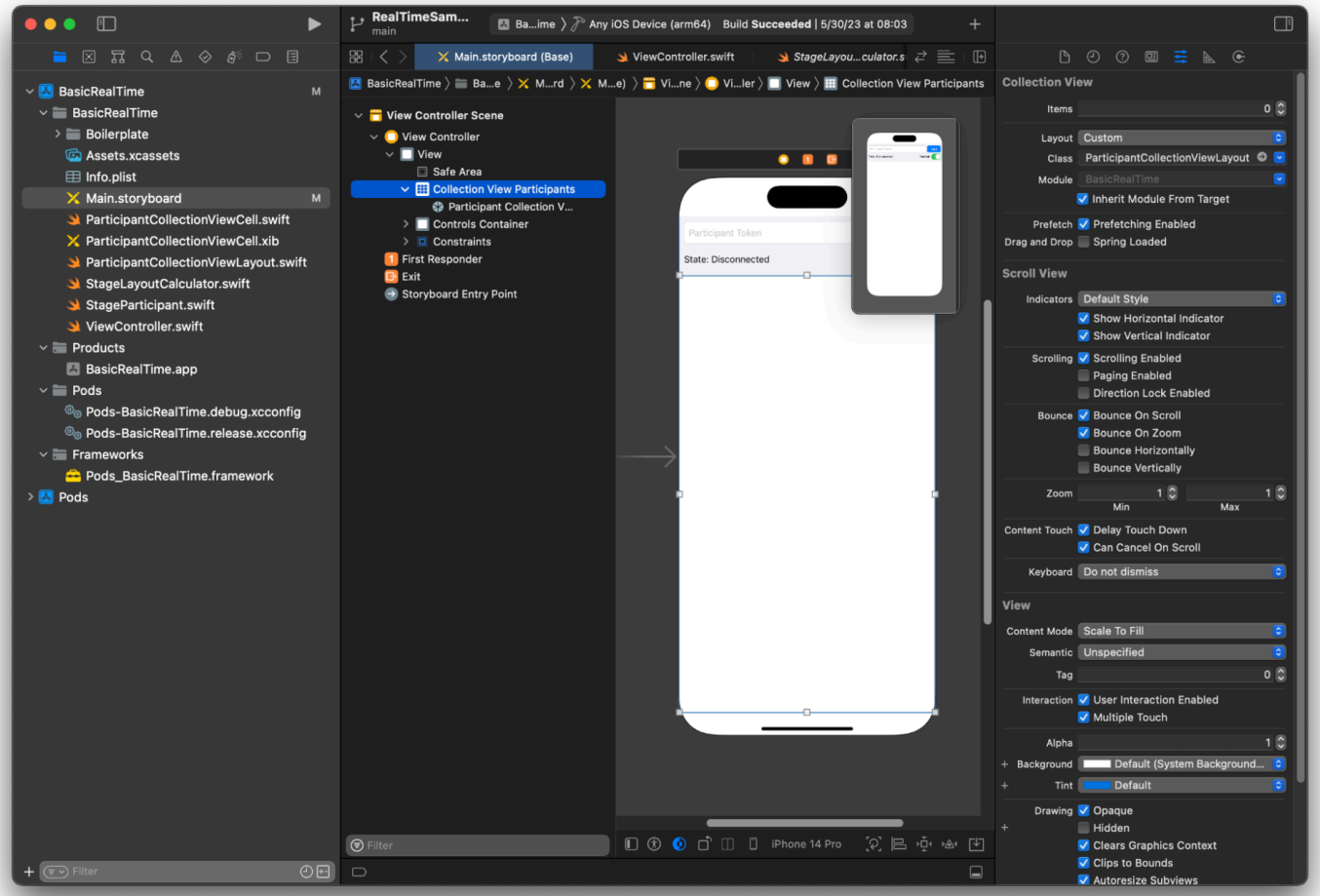
les colonnes peuvent avoir des largeurs différentes par ligne. Voir le commentaire de code au-dessus de la variable `layouts` pour une description de la façon de personnaliser ce comportement.

```
import Foundation
import UIKit

class StageLayoutCalculator {

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

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

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

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

        let halfPadding = padding / 2.0

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

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

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

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

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

}
```

De retour à `Main.storyboard`, veillez à utiliser la classe que nous venons de créer pour définir la classe de disposition de `UICollectionView` :



Connexion des actions de l’interface utilisateur

Nous nous rapprochons, il nous reste quelques `IBActions` à créer.

Nous allons d’abord gérer le bouton Rejoindre. Il répond différemment en fonction de la valeur de `connectingOrConnected`. Lorsqu’il est déjà connecté, il a pour effet de quitter la scène. S’il est déconnecté, il lit le texte du jeton `UITextField` et crée un `IVSStage` avec ce texte. Ensuite, nous ajoutons notre `ViewController` en tant que `strategy`, `errorDelegate`, et moteur de rendu pour `IVSStage`, et enfin nous rejoignons la scène de manière asynchrone.

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

L'autre action de l'interface utilisateur que nous devons connecter est le commutateur de publication :

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

Affichage des participants

Enfin, nous devons rendre les données que nous recevons du SDK sur la cellule de participant que nous avons créée précédemment. La logique du `UICollectionView` est déjà terminée, il ne nous reste plus qu'à implémenter l'API `set` dans `ParticipantCollectionViewCell.swift`.

Nous allons commencer par ajouter la fonction `empty`, puis nous l'étudierons étape par étape :

```
func set(participant: StageParticipant) {

}
```

Nous gérons d'abord l'état simplifié, l'identifiant du participant, l'état de publication et l'état d'abonnement. Pour ceux-ci, nous mettons simplement à jour nos `UILabels` directement :

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

Les propriétés de texte des énumérations de publication et d'abonnement proviennent d'extensions locales :

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

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

Ensuite, nous mettons à jour les états de coupure de l'audio et de la vidéo. Pour obtenir les états de coupure, nous devons trouver `IVSImageDevice` et `IVSAudioDevice` à partir du tableau de `streams`. Pour optimiser les performances, nous mémorisons les derniers identifiants des appareils connectés.

```
// 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)"
```

Enfin, nous voulons afficher un aperçu du `imageDevice` et affichez les statistiques audio du `audioDevice` :

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

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

La dernière fonction que nous devons créer est `updatePreview()`, qui ajoute un aperçu du participant à notre vue :

```
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)
        }
    }
}
```

Ce qui précède utilise une fonction d'assistance sur `UIView` pour faciliter l'intégration de sous-vues :

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

Surveillance du streaming en temps réel Amazon IVS

Qu’est-ce qu’une session d’étape ?

Une session d'étape commence lorsque le premier participant rejoint une étape et se termine quelques minutes après que le dernier participant cesse d'être diffusé sur l'étape. Les sessions d'étape facilitent le débogage des étapes de longue durée en séparant les événements et les participants en étapes de courte durée.

Afficher les sessions d'étape et les participants

Instructions de la console

1. Ouvrez la console Amazon IVS.

   (Vous pouvez également accéder à la console Amazon IVS via la console de gestion AWS.)
2. Dans le panneau de navigation, choisissez Étapes. (Si le volet de navigation est réduit, ouvrez-le d’abord en choisissant l’icône en forme de hamburger.)
3. Choisissez une étape pour accéder à sa page de détails.
4. Faites défiler la page vers le bas jusqu’à la section Sessions d’étape, puis sélectionnez une session d’étape pour afficher sa page de détails.
5. Pour afficher les participants à la session, faites défiler la page vers le bas jusqu’à la section Participants, puis sélectionnez un participant pour afficher sa page de détails, y compris des tableaux pour les métriques Amazon CloudWatch.

Afficher les événements pour un participant

Les événements sont envoyés lorsque le statut d'un participant change dans une scène, par exemple lorsqu'il rejoint une scène ou qu'il rencontre une erreur lors de la tentative de publication sur une scène. Les erreurs ne provoquent pas toutes des événements. Par exemple, les erreurs réseau côté client et les erreurs de signature de jeton ne sont pas envoyées en tant qu'événements. Pour gérer ces erreurs dans votre application cliente, utilisez les kits SDK de diffusion IVS.

Instructions de la console

1. Accédez à la page de détails du participant en suivant les instructions ci-dessus.
2. Faites défiler la page vers le bas jusqu'à la section Événements. Cette section affiche une liste ordonnée des événements des participants. Consultez la section Utilisation d'Amazon EventBridge avec Amazon IVS pour en savoir plus sur les événements émis pour les participants.

Instructions de la CLI

L'accès aux événements des sessions de scène à partir de l'AWS CLI est une option avancée. Vous devez d'abord télécharger et configurer la CLI sur votre machine. Pour plus de détails, consultez le Guide de l'utilisateur de l'Interface de ligne de commande AWS.

1. Répertoriez les sessions d'étape pour trouver une session d'étape :

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

2. Répertoriez les participants à une session d'étape pour trouver un participant :

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

3. Répertoriez les événements relatifs à une session d'étape et à un participant :

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

Voici un exemple de réponse à l'appel `list-participant-events` :

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

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

Accès aux métriques CloudWatch

Pour que les métriques CloudWatch soient disponibles, les versions du SDK de diffusion IVS sont requises : Web 1.5.0 ou une version ultérieure, Android 1.12.0 ou une version ultérieure ou iOS 1.12.0 ou une version ultérieure.

Instructions pour la console CloudWatch

1. Ouvrez la console CloudWatch à l'adresse https://console.aws.amazon.com/cloudwatch/.
2. Dans le panneau latéral de navigation, développez le menu déroulant Metrics (Métriques), puis sélectionnez All metrics (Toutes les métriques).
3. Sous l'onglet Parcourir, à l'aide de la liste déroulante sans étiquette à gauche, sélectionnez votre région « d'accueil » dans laquelle votre ou vos canaux ont été créés. Pour en savoir plus sur les régions, consultez Solution mondiale, contrôle régional. Pour obtenir une liste des régions prises en charge, consultez la page Amazon IVS dans les Références générales AWS.
4. Au bas de l'onglet Parcourir, sélectionnez l'espace de noms IVSRealTime.
5. Effectuez l'une des actions suivantes :
   a. Dans la barre de recherche, entrez votre ID de ressource (partie de l'ARN, `arn:::ivs:stage/<resource id>`).

      Sélectionnez ensuite IVSRealTime > Métriques d'étape.

b. Si IVSRealTime apparaît comme un service sélectionnable sous Espaces de noms AWS, sélectionnez-le. Il sera répertorié si vous utilisez le Streaming en temps réel Amazon IVS et qu'il envoie des métriques à Amazon CloudWatch. (Si IVSRealTime n'est pas répertorié, vous ne disposez pas de métriques Amazon IVS.)

   Choisissez ensuite un groupe de dimensions comme vous le souhaitez ; les dimensions disponibles sont répertoriées dans les Métriques CloudWatch ci-dessous.

6. Choisissez des métriques pour ajouter au graphique. Les métriques disponibles sont répertoriées dans les Métriques CloudWatch ci-dessous.

Vous pouvez également accéder au graphique CloudWatch de votre session de flux de streaming à partir de la page de détails de ladite session, en sélectionnant le bouton View in CloudWatch (Afficher dans CloudWatch).

Instructions de la CLI

Vous pouvez également accéder aux métriques à l'aide de l'AWS CLI. Pour cela, vous devez d'abord télécharger et configurer la CLI sur votre machine. Pour plus de détails, consultez le Guide de l'utilisateur de l'Interface de ligne de commande AWS.

Ensuite, pour accéder aux métriques de streaming en temps réel Amazon IVS à l'aide de l'AWS CLI :

- À partir d'une invite de commande, exécutez :

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

Pour de plus amples informations, consultez Utilisation des métriques Amazon CloudWatch dans le Guide de l'utilisateur Amazon CloudWatch.

Métriques CloudWatch : streaming en temps réel IVS

Amazon IVS fournit les métriques suivantes dans l'espace de noms AWS/IVSRealTime.

Pour que les métriques CloudWatch soient disponibles, vous devez utiliser le SDK de diffusion Web 1.5.2 ou une version ultérieure.

La dimension peut avoir les valeurs valides suivantes :

- La dimension `Stage` est un ID de ressource (partie de l'ARN, `arn:::stage/<resource id>`).
- La dimension `Participant` est un `participantID`.
- La valeur pour `SimulcastLayer` est « élevé », « moyen », « faible » ou « sans portée » pour un `MediaType` de « vidéo » ou « désactivé » pour un `MediaType` de « audio ». Cette valeur peut également être vide.
- La dimension `MediaType` est « vidéo » ou « audio » (chaîne).

Métrique	Dimension	Description
`DownloadPacketLoss`	`Stage`	Chaque exemple représente le pourcentage de paquets perdus par un abonné donné au cours d'un télécharg ement depuis le serveur IVS. Unité : pourcentage Statistiques valides : Moyenne, Maximum, Minimum – Débit moyen, maximum et minimum (respectivement) de paquets perdus sur l'intervalle configuré
`DownloadPacketLoss`	`Stage,Participant`	Filtre `DownloadPacketLoss` par participant, pour les abonnés qui sont également des diffuseurs de publication. Les exemples représentent le pourcentage de paquets perdus par l'abonné au cours d'un télécharg ement depuis le serveur IVS. Les exemples ne sont émis que lorsque le participant est également un diffuseur de publication. Unité : pourcentage Statistiques valides : Moyenne, Maximum, Minimum – Débit moyen, maximum et minimum (respectivement) d'images abandonnées sur l'intervalle configuré
`DroppedFrames`	`Stage`	Chaque échantillon représente le pourcentage d'images abandonnées par un abonné donné. Unité : pourcentage

Métrique	Dimension	Description
		Statistiques valides : Moyenne, Maximum, Minimum – Débit moyen, maximum et minimum (respectivement) d'images abandonnées sur l'intervalle configuré
`DroppedFrames`	`Stage,Participant`	Filtre `DroppedFrames` par participant, pour les abonnés qui sont également des diffuseurs de publication. Les exemples représentent le pourcentage d'images perdues entre le participant abonné et tous les diffuseurs de publication participant à l'étape. Les exemples ne sont émis que lorsque le participant est également un diffuseur de publication. Unité : pourcentage Statistiques valides : Moyenne, Maximum, Minimum – Débit moyen, maximum et minimum (respectivement) d'images abandonnées sur l'intervalle configuré
`PublishBitrate`	`Stage`	Les échantillons émis représentent le débit total auquel un éditeur donné envoie des données vidéo et audio (additionnées pour toutes les couches de diffusion simulcast). Unité : bits par seconde Statistiques valides : Moyenne, Maximum, Minimum – Débit binaire moyen, maximum et minimum (respectivement) au cours de l'intervalle configuré

Métrique	Dimension	Description
`PublishBitrate`	`Stage, Participant, SimulcastLayer, MediaType`	Filtres `PublishBitrate` par participant, couche simulcast et type de média. L'ID de couche simulcast est défini par le kit SDK de diffusion. Lorsque simulcast est désactivé, cet ID de couche sera défini sur « désactivé ». Le type de média est vidéo ou audio. Unité : bits par seconde Statistiques valides : Moyenne, Maximum, Minimum – Débit binaire moyen, maximum et minimum (respectivement) au cours de l'intervalle configuré
`Publishers`	`Stage`	Nombre de participants publiant sur l'étape. Unité : nombre Statistiques valides : Moyenne, Maximum, Minimum
`PublishResolution`	`Stage, Participant, SimulcastLayer, MediaType`	Nombre de pixels sur la plus petite des valeurs entre la largeur et la hauteur de l’image. Par exemple, pour une image paysage de 1920 x 1080, la résolution de publication est 1080. Pour une image portrait de 720 x 1280, la résolution de publication est 720. Unité : nombre Statistiques valides : Moyenne, Maximum, Minimum
`SubscribeBitrate`	`Stage`	Les échantillons émis représentent le débit total auquel un abonné donné reçoit des données vidéo et audio. Unité : bits par seconde Statistiques valides : Moyenne, Maximum, Minimum – Débit binaire moyen, maximum et minimum (respectivement) au cours de l’intervalle configuré

Métrique	Dimension	Description
`SubscribeBitrate`	`Stage,Participant,MediaType`	Filtre `SubscribeBitrate` par participant, pour les abonnés qui sont également des diffuseurs de publication. Les exemples représentent le débit auquel un abonné donné reçoit le `MediaType` donné. Les exemples ne sont émis que lorsque le participant abonné effectue une publication. Unité : bits par seconde Statistiques valides : Moyenne, Maximum, Minimum – Débit binaire moyen, maximum et minimum (respectivement) au cours de l'intervalle configuré
`Subscribers`	`Stage`	Nombre de participants abonnés à l'étape. Notez que les participants qui publient et s'abonnent activement sont considérés à la fois comme éditeurs et comme abonnés. Unité : nombre Statistiques valides : Moyenne, Maximum, Minimum

SDK de diffusion IVS (Streaming en temps réel)

Le SDK de diffusion par streaming en temps réel Amazon Interactive Video Services (IVS) est destiné aux développeurs qui créent des applications avec Amazon IVS. Ce kit SDK est conçu pour tirer parti de l'architecture Amazon IVS et nous y apporterons continuellement des améliorations et de nouvelles fonctionnalités, en plus d'Amazon IVS. En tant que kit SDK de diffusion natif, il est conçu pour minimiser l'impact sur les performances de votre application et sur les périphériques avec lesquels vos utilisateurs accèdent à votre application.

Notez que le SDK de diffusion est utilisé à la fois pour envoyer et recevoir des vidéos ; c'est-à-dire que vous utilisez le même SDK pour les hôtes et les spectateurs. Aucun SDK de lecteur distinct n'est nécessaire.

Votre application peut tirer parti des fonctionnalités clés du kit SDK de diffusion Amazon IVS :

- Streaming haute qualité : le kit SDK de diffusion prend en charge le streaming haute qualité. Capturez des vidéos à partir de votre caméra et encodez-les jusqu'à 720p.
- Ajustements automatiques du débit binaire : les utilisateurs de smartphones sont mobiles, de sorte que les conditions de leur réseau peuvent changer tout au long de la diffusion. Le kit SDK de diffusion Amazon IVS ajuste automatiquement le débit binaire de la vidéo pour s'adapter aux conditions changeantes du réseau.
- Support des formats portrait et paysage : quelle que soit la façon dont vos utilisateurs tiennent leurs appareils, l'image apparaît dans le bon sens et à la bonne échelle. Le kit SDK de diffusion prend en charge les tailles de canevas portrait et paysage. Il gère automatiquement les proportions lorsque les utilisateurs font pivoter leur appareil et quittent l'orientation configurée.
- Streaming sécurisé : les diffusions de vos utilisateurs sont chiffrées à l'aide de TLS ; ils peuvent donc sécuriser leurs flux.
- Périphériques audio externes : le kit SDK de diffusion Amazon IVS prend en charge les microphones externes à prise audio, USB et Bluetooth SCO.

Exigences de la plateforme

Plateformes natives

Plateforme	Versions prises en charge
Android	9.0 et versions ultérieures : notez que les clients peuvent créer avec la version 5.0 mais ne pourront pas utiliser la fonctionnalité de diffusion en temps réel.
iOS	14 et versions ultérieures

IVS prend en charge au moins 4 versions majeures d'iOS et 6 versions majeures d'Android. Notre prise en charge des versions actuelles peut s'étendre au-delà de ces minimums. Si une version majeure n'est plus prise en charge, les clients seront informés par des notes de mise à jour du SDK au moins 3 mois à l'avance.

Navigateurs de bureau

Navigateur	Plateformes prises en charge	Versions prises en charge
Chrome	Windows, macOS	Deux versions principales (la version actuelle et la version la plus récente)
Firefox	Windows, macOS	Deux versions principales (la version actuelle et la version la plus récente)
Edge	(Windows 8.1 et versions ultérieures)	Deux versions principales (la version actuelle et la version la plus récente) Exclut Edge Legacy
Safari	macOS	Deux versions principales (la version actuelle et la version la plus récente)

Navigateurs mobiles (iOS et Android)

Navigateur	Plateformes prises en charge	Versions prises en charge
Chrome	iOS, Android	Deux versions principales (la version actuelle et la version la plus récente)
Firefox	Android	Deux versions principales (la version actuelle et la version la plus récente)
Safari	iOS	Deux versions principales (la version actuelle et la version la plus récente)

Limitations connues

- Sur tous les appareils mobiles, nous vous déconseillons de diffuser ou de vous abonner à quatre participants ou plus en même temps. Cela pourrait entrainer des problèmes liés aux artefacts vidéo et aux écrans noirs. Si vous avez besoin de plus de participants, configurez la diffusion et l'abonnement audio uniquement.
- Nous vous déconseillons de composer une scène et de la diffuser sur une chaîne avec Android Mobile Web. Cela pourrait réduire les performances et provoquer d'éventuels plantage. Si une fonctionnalité de diffusion est requise, intégrez le SDK de diffusion Android par diffusion en temps réel IVS.

Webview

Le SDK de diffusion Web ne prend pas en charge les Webview ou les environnements similaires au Web (téléviseurs, consoles, etc.). Pour les implémentations mobiles, consultez le guide du SDK de diffusion par streaming en temps réel pour Android et pour iOS.

Accès requis à l'appareil

Le kit SDK de diffusion nécessite l'accès aux caméras et microphones de l'appareil, à la fois ceux intégrés à l'appareil et ceux connectés via Bluetooth, USB ou prise audio.

Support

Remarque : le SDK de diffusion est en constante amélioration. Consultez la rubrique Notes de mise à jour Amazon IVS pour connaître les versions disponibles et les problèmes résolus. Le cas échéant, avant de contacter le support technique, mettez à jour la version du kit SDK de diffusion et vérifiez si cela résout votre problème.

Contrôle de version

Les kits SDK de diffusion Amazon IVS utilisent la gestion sémantique des versions.

Pour ce sujet, supposons que :

- la dernière version est la version 4.1.3 ;
- la dernière version de la version majeure précédente est la version 3.2.4 ;
- la dernière version de la version 1.x est la version 1.5.6.

De nouvelles fonctions rétrocompatibles sont ajoutées en tant que versions mineures de la dernière version. Dans ce cas, la prochaine série de nouvelles fonctions sera ajoutée dans la version 4.2.0.

Des corrections de bogues mineurs rétrocompatibles sont ajoutées en tant que versions de correctifs de la dernière version. Ici, la prochaine série de corrections de bogues mineurs sera ajoutée en tant que version 4.1.4.

Les corrections de bogues majeurs rétrocompatibles sont traitées différemment. Elles sont ajoutées à plusieurs versions :

- Version de correctifs de la dernière version. Ici, il s’agit de la version 4.1.4.
- Version de correctifs de la version mineure précédente. Ici, il s’agit de la version 3.2.5.
- Version de correctifs de la dernière version 1.x. Ici, il s’agit de la version 1.5.7.

Les principales corrections de bogues sont définies par l’équipe produit d’Amazon IVS. Des exemples typiques sont les mises à jour de sécurité critiques et d’autres correctifs nécessaires pour les clients.

Remarque : dans les exemples ci-dessus, les versions publiées s’incrémentent sans ignorer de numéros (par exemple, de 4.1.3 à 4.1.4). En réalité, un ou plusieurs numéros de correctifs peuvent rester internes et ne pas être publiés, de sorte que la version publiée peut s’incrémenter de 4.1.3 à 4.1.6, par exemple.

SDK de diffusion IVS : guide pour le Web (Streaming en temps réel)

Le SDK de diffusion Web par streaming en temps réel IVS fournit aux développeurs les outils nécessaires pour créer des expériences interactives en temps réel sur le Web. Ce kit SDK est destiné aux développeurs qui créent des applications web avec Amazon IVS.

Le SDK de diffusion Web permet aux participants d'envoyer et de recevoir des vidéos. Le SDK prend en charge les opérations suivantes :

- Rejoindre une étape
- Publier du contenu multimédia à l'intention des autres participants de l'étape
- S'abonner à du contenu multimédia d'autres participants de l'étape
- Gérer et surveiller la vidéo et le son publiés sur l'étape
- Obtenir des statistiques WebRTC pour chaque connexion d'appairage
- Toutes les opérations à partir du SDK de diffusion Web par streaming à faible latence

Dernière version du SDK de diffusion Web : 1.8.0 (notes de version)

Documentation de référence : pour plus d'informations sur les méthodes les plus importantes disponibles dans le SDK Amazon IVS Web Broadcast, consultez https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference. Assurez-vous que la version la plus récente du kit SDK est sélectionnée.

Exemple de code : les exemples ci-dessous constituent un bon point de départ pour commencer à utiliser rapidement le kit SDK :

- HTML et JavaScript
- React

Exigences de la plateforme : consultez le SDK de diffusion Amazon IVS pour obtenir la liste des plateformes prises en charge

Démarrage

Importations

Les composantes de base temps réel se trouvent dans un espace de noms différent de celui des modules de diffusion racine.

Utilisation d’une balise de script

En utilisant les mêmes importations de scripts, les classes et les énumérations définies dans les exemples ci-dessous se trouvent sur l’objet global `IVSBroadcastClient` :

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

Utilisation de npm

Les classes, les énumérations et les types peuvent également être importés depuis le module de package :

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

Demander des autorisations

Votre application doit demander l’autorisation d’accéder à la caméra et au microphone de l’utilisateur, et cela doit être réalisé en utilisant HTTPS. (Ce n’est pas spécifique à Amazon IVS ; cette autorisation est requise pour toute application devant accéder aux caméras et aux microphones.)

Voici un exemple de fonction qui montre comment demander et capturer des autorisations pour les périphériques audio et vidéo :

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

Pour plus d'informations, consultez l'API Permissions et MediaDevices. getUserMedia().

Répertorier les périphériques disponibles

Pour voir quels appareils peuvent être capturés, interrogez la méthode MediaDevices.enumerateDevices () du navigateur :

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

Récupérer un MediaStream depuis un appareil

Après avoir acquis la liste des périphériques disponibles, vous pouvez récupérer un flux à partir d'un nombre quelconque de périphériques. Par exemple, vous pouvez utiliser la méthode `getUserMedia()` pour récupérer un flux d'une caméra.

Si vous souhaitez spécifier le périphérique à partir duquel vous souhaitez capturer le flux, vous pouvez définir explicitement le `deviceId` dans la section `audio` ou `video` des contraintes multimédias. Vous pouvez également omettre le `deviceId` et demander aux utilisateurs de sélectionner leurs périphériques à l'invite du navigateur.

Vous pouvez également spécifier une résolution de caméra idéale à l'aide des contraintes `width` et `height`. (Pour en savoir plus sur ces contraintes, cliquez ici.) Le kit SDK applique automatiquement des contraintes de largeur et de hauteur qui correspondent à votre résolution de diffusion maximale ; cependant, il est conseillé de les appliquer vous-même pour vous assurer que le rapport hauteur/largeur de la source n'est pas modifié une fois que vous avez ajouté la source au kit SDK.

Pour le streaming en temps réel, assurez-vous que le contenu multimédia est limité à une résolution de 720p. Plus précisément, vos valeurs `getUserMedia` et celles des `getDisplayMedia` contraintes pour la largeur et la hauteur ne doivent pas dépasser 921600 (1280* 720) lorsqu'elles sont multipliées ensemble.

```
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 },
});
```

Publication et abonnement

Concepts

Trois concepts de base sous-tendent la fonctionnalité temps réel : scène, stratégie et événements. L'objectif de la conception consiste à minimiser la quantité de logique côté client nécessaire à la création d'un produit fonctionnel.

Étape

La classe `Stage` est le principal point d'interaction entre l'application hôte et le kit SDK. Il représente l'étape elle-même et est utilisé pour rejoindre et quitter l'étape. La création et la participation à une étape nécessitent une chaîne de jetons valide et non expirée provenant du plan de contrôle (représentée par `token`). Il est très facile de rejoindre une étape et de la quitter :

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

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

stage.leave();
```

Strategy

L'interface `StageStrategy` permet à l'application hôte de communiquer l'état de l'étape souhaité au kit SDK. Trois fonctions doivent être mises en œuvre : `shouldSubscribeToParticipant`, `shouldPublishParticipant` et `stageStreamsToPublish`. Elles sont toutes abordées ci-dessous.

Pour utiliser une stratégie définie, transmettez-la au constructeur `Stage`. Voici un exemple complet d'application utilisant une stratégie pour publier la webcam d'un participant sur l'étape et s'abonner à tous les participants. L'objectif de chaque fonction de stratégie requise est expliqué en détail dans les sections suivantes.

```
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();
```

Abonnement aux participants

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

Lorsqu'un participant distant rejoint l'étape, le kit SDK interroge l'application hôte sur l'état de l'abonnement souhaité pour ce participant. Les options sont `NONE`, `AUDIO_ONLY` et `AUDIO_VIDEO`. Lors d'un renvoi d'une valeur pour cette fonction, l'application hôte n'a pas à se soucier de l'état de publication, de l'état actuel de l'abonnement ou de l'état de la connexion de l'étape. Si `AUDIO_VIDEO` est renvoyé, le kit SDK attend que le participant distant effectue une publication avant de s'abonner, puis il met à jour l'application hôte en émettant des événements tout au long du processus.

Voici un exemple d'implémentation :

```
const strategy = {

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

```
   }

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

Il s'agit de l'implémentation complète de cette fonction pour une application hôte qui souhaite toujours que tous les participants se voient mutuellement, comme une application de chat vidéo.

Des implémentations plus avancées sont également possibles. Utilisez la propriété `userInfo` sur `ParticipantInfo` pour vous abonner de manière sélective aux participants en fonction des attributs fournis par le serveur :

```
const strategy = {

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

Elle peut servir à créer une scène où les modérateurs peuvent surveiller tous les invités sans être vus ou entendus. L'application hôte pourrait utiliser une logique métier supplémentaire pour permettre aux modérateurs de se voir mutuellement tout en restant invisible pour les invités.

Publication

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

Une fois connecté à l'étape, le kit SDK interroge l'application hôte pour savoir si un participant en particulier doit effectuer une publication. Elle n'est invoquée que pour les participants locaux autorisés à publier sur la base du jeton fourni.

Voici un exemple d'implémentation :

```
const strategy = {
```

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

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

Il s'agit d'une application de chat vidéo standard dans laquelle les utilisateurs souhaitent toujours publier. Ils peuvent désactiver et réactiver leur son et leur vidéo pour être instantanément masqués ou vus/entendus. (Ils peuvent également utiliser la fonction de publication/d'annulation de la publication, mais c'est beaucoup plus lent. Il est préférable de désactiver ou de réactiver le son dans les cas d'utilisation où il est souvent souhaitable de modifier la visibilité.)

Choix des flux à publier

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

Lors de la publication, cette fonction est utilisée pour déterminer les flux audio et vidéo à publier. Ce point est abordé plus en détail dans la section Publier un flux multimédia.

Mise à jour de la stratégie

La stratégie se veut dynamique : les valeurs renvoyées par n'importe laquelle des fonctions ci-dessus peuvent être modifiées à tout moment. Par exemple, si l'application hôte ne souhaite pas publier tant que l'utilisateur final n'a pas appuyé sur un bouton, vous pouvez renvoyer une variable depuis `shouldPublishParticipant` (quelque chose comme `hasUserTappedPublishButton`). Lorsque cette variable change en fonction d'une interaction de l'utilisateur final, appelez `stage.refreshStrategy()` pour signaler au kit SDK qu'il doit interroger la stratégie pour connaître les dernières valeurs, en appliquant uniquement les éléments qui ont changé. Si le kit SDK constate que la valeur `shouldPublishParticipant` a changé, il lance le processus de publication. Si les requêtes du kit SDK et toutes les fonctions renvoient la même valeur qu'auparavant, l'appel `refreshStrategy` ne modifie pas l'étape.

Si la valeur de retour de `shouldSubscribeToParticipant` passe de `AUDIO_VIDEO` à `AUDIO_ONLY`, le flux vidéo est supprimé pour tous les participants dont les valeurs renvoyées ont été modifiées, s'il existait déjà un flux vidéo.

En général, l'étape utilise la stratégie pour appliquer au mieux la différence entre les stratégies précédentes et actuelles, sans que l'application hôte n'ait à se soucier de tout l'état requis

pour la gérer correctement. Pour cette raison, et pour réduire les frais, envisagez d'appeler `stage.refreshStrategy()`, car cela ne fait rien à moins que la stratégie ne change.

Événements

Une instance `Stage` est un émetteur d'événements. En utilisant `stage.on()`, l'état de l'étape est communiqué à l'application hôte. Les mises à jour de l'interface utilisateur de l'application hôte peuvent généralement être entièrement prises en charge par les événements. Les événements sont les suivants :

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

Pour la plupart de ces événements, les `ParticipantInfo` correspondantes sont fournies.

Les informations fournies par les événements ne devraient pas avoir d'impact sur les valeurs de retour de la stratégie. Par exemple, la valeur de retour de `shouldSubscribeToParticipant` ne devrait pas changer lors de l'appel de `STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`. Si l'application hôte souhaite s'abonner à un participant en particulier, elle doit renvoyer le type d'abonnement souhaité, quel que soit l'état de publication de ce participant. Le kit SDK est chargé de s'assurer que l'état souhaité de la stratégie est appliqué au bon moment en fonction de l'état de l'étape.

Publier un flux multimédia

Les appareils locaux tels que les microphones et les caméras sont récupérés en suivant les mêmes étapes que celles décrites ci-dessus dans la section Récupérer un appareil MediaStream depuis un appareil. Dans l'exemple, nous utilisons`MediaStream` pour créer une liste d'objets `LocalStageStream` utilisés à des fins de publication par le kit SDK :

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

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

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

Publier un partage d'écran

Les applications doivent souvent publier un partage d'écran en plus de la caméra Web de l'utilisateur. La publication d'un partage d'écran nécessite la création d'une `Stage` supplémentaire dotée de son propre jeton unique.

```
// 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();
```

Afficher et supprimer des participants

Une fois l'inscription terminée, vous recevez un tableau d'objets `StageStream` via l'événement `STAGE_PARTICIPANT_STREAMS_ADDED`. L'événement vous fournit également des informations sur les participants pour vous aider lors de l'affichage des flux de contenu multimédia :

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

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

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

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

Lorsqu'un participant arrête de publier ou en cas de désabonnement d'un flux, la fonction `STAGE_PARTICIPANT_STREAMS_REMOVED` est appelée avec les flux qui ont été supprimés. Les applications hôte doivent utiliser ce signal pour supprimer le flux vidéo du participant du DOM.

`STAGE_PARTICIPANT_STREAMS_REMOVED` est invoqué pour tous les scénarios dans lesquels un flux peut être supprimé, notamment :

- Le participant distant arrête de publier.
- Un appareil local se désabonne ou modifie l'abonnement de `AUDIO_VIDEO` en `AUDIO_ONLY`.
- Le participant distant quitte l'étape.
- Le participant local quitte l'étape.

Comme `STAGE_PARTICIPANT_STREAMS_REMOVED` est invoqué pour tous les scénarios, aucune logique métier personnalisée n'est requise pour supprimer des participants de l'interface utilisateur lors des opérations de départ locales ou à distance.

Désactiver et réactiver le son des flux de médias sociaux

Les objets `LocalStageStream` ont une fonction `setMuted` qui contrôle si le son du flux est désactivé. Cette fonction peut être appelée sur le flux avant ou après son renvoi par la fonction de stratégie `stageStreamsToPublish`.

Important : si une nouvelle instance d'objet `LocalStageStream` est renvoyée par `stageStreamsToPublish` après un appel à `refreshStrategy`, l'état muet du nouvel objet de flux est appliqué à l'étape. Lorsque vous créez des instances `LocalStageStream`, veillez à ce que l'état muet attendu soit conservé.

Surveiller l'état muet du contenu multimédia des participants distants

Lorsque les participants modifient l'état muet de leur vidéo ou audio, l'événement `STAGE_STREAM_MUTE_CHANGED` est déclenché avec une liste des flux qui ont changé. Utilisez la propriété `isMuted` sur `StageStream` pour mettre à jour votre interface utilisateur en conséquence :

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

Vous pouvez également consulter les informations StageParticipantInfod'état indiquant si le son ou la vidéo est désactivé :

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

Obtenir les statistiques WebRTC

Pour obtenir les dernières statistiques WebRTC relatives à un flux de publication ou d'abonnement, utilisez `getStats` sur `StageStream`. Il s'agit d'une méthode asynchrone avec laquelle vous pouvez récupérer des statistiques soit via une attente, soit en enchaînant une promesse. Le résultat est un `RTCStatsReport`, un dictionnaire contenant toutes les statistiques standard.

```
try {
   const stats = await stream.getStats();
```

```
} catch (error) {
   // Unable to retrieve stats
}
```

Optimisation de contenu multimédia

Pour des performances optimales, il est recommandé de limiter les appels `getUserMedia` et `getDisplayMedia` aux contraintes suivantes :

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

Vous pouvez restreindre davantage le contenu multimédia à l'aide d'options supplémentaires transmises au constructeur `LocalStageStream` :

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

Dans le code ci-dessus :

- `minBitrate` définit le débit minimum que le navigateur doit être censé utiliser. Toutefois, un flux vidéo de faible complexité peut pousser l'encodeur à descendre en dessous de ce débit.
- `maxBitrate` définit un débit maximal que le navigateur ne doit pas dépasser pour ce flux.
- `maxFramerate` définit une fréquence d'images maximale que le navigateur ne doit pas dépasser pour ce flux.
- L'option `simulcast` n'est utilisable que sur les navigateurs basés sur Chromium. Elle permet d'envoyer trois couches de rendu du flux.

- Cela permet au serveur de choisir le rendu à envoyer aux autres participants, en fonction des limites de leur réseau.
- Lorsque `simulcast` est spécifié en même temps qu'une valeur `maxBitrate` et/ou `maxFramerate`, on s'attend à ce que la couche de rendu la plus élevée soit configurée en tenant compte de ces valeurs, à condition que `maxBitrate` ne descende pas en dessous de la valeur `maxBitrate` par défaut de 900 kbps de la deuxième couche la plus élevée du SDK interne.
- Si `maxBitrate` est spécifié comme étant trop faible par rapport à la valeur par défaut de la deuxième couche la plus élevée, `simulcast` sera désactivé.
- `simulcast` ne peut pas être activé et désactivé sans republier le média. Pour ce faire, il faut que `shouldPublishParticipant` renvoie la valeur `false`, que `refreshStrategy` soit appelé, que `shouldPublishParticipant` renvoie la valeur `true` et que `refreshStrategy` soit à nouveau appelé.

Obtenir les attributs des participants

Si vous spécifiez des attributs dans la demande de point de terminaison `CreateParticipantToken`, vous pouvez les voir dans les propriétés `StageParticipantInfo` :

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

Gestion des problèmes de réseau

En cas de perte de la connexion réseau de l'appareil local, le kit SDK essaie de se reconnecter en interne sans aucune action de l'utilisateur. Dans certains cas, le kit SDK échoue et une action de l'utilisateur est requise.

D'une manière générale, l'état de l'étape peut être géré via l'événement `STAGE_CONNECTION_STATE_CHANGED` :

```
stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
   switch (state) {
      case StageConnectionState.DISCONNECTED:
         // handle disconnected UI
         break;
      case StageConnectionState.CONNECTING:
```

```
         // handle establishing connection UI
         break;
      case StageConnectionState.CONNECTED:
         // SDK is connected to the Stage
         break;
      case StageConnectionState.ERRORED:
         // unrecoverable error detected, please re-instantiate
         Break;
})
```

En général, le fait de rencontrer des erreurs après avoir correctement rejoint une étape indique que le kit SDK a perdu la connexion et n'a pas réussi à la rétablir. Créez un objet `Stage` et essayez de le joindre lorsque les conditions du réseau s'améliorent.

Diffuser la scène sur un canal IVS

Pour diffuser une étape, créez une session `IVSBroadcastClient` distincte, puis suivez les instructions habituelles de diffusion avec le kit SDK, décrites ci-dessus. La liste des `StageStream` exposés via `STAGE_PARTICIPANT_STREAMS_ADDED` peut être utilisée pour récupérer les flux de contenu multimédia des participants, qui peuvent être appliqués à la composition du flux de diffusion, comme suit :

```
// 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;
        }
```

```
    })
})
```

Vous pouvez éventuellement composer une scène et la diffuser sur un canal IVS à faible latence, afin de toucher un public plus large. Consultez la section Activation d'hôtes multiples sur un flux Amazon IVS dans le guide de l'utilisateur du streaming à faible latence IVS.

Problèmes connus et solutions de contournement

- Lorsque vous fermez les onglets du navigateur ou quittez des navigateurs sans appeler `stage.leave()`, des utilisateurs peuvent toujours apparaître dans la session avec un cadre figé ou un écran noir pendant maximum 10 secondes.

  Solution de contournement : aucune.

- Les sessions Safari apparaissent par intermittence avec un écran noir pour les utilisateurs qui rejoignent une session en cours.

  Solution de contournement : actualisez le navigateur et reconnectez la session.

- Safari ne se rétablit pas correctement après un changement de réseau.

  Solution de contournement : actualisez le navigateur et reconnectez la session.

- La console du développeur répète une erreur `Error: UnintentionalError at StageSocket.onClose`.

  Solution de contournement : il n'est possible de créer qu'une seule étape par jeton de participant. Cette erreur se produit lorsque plusieurs instances `Stage` sont créées avec le même jeton de participant, que l'instance se trouve sur un ou plusieurs appareils.

- Il se peut que vous éprouviez des difficultés à maintenir un `StageParticipantPublishState.PUBLISHED` état et que vous receviez `StageParticipantPublishState.ATTEMPTING_PUBLISH` des états répétés lorsque vous écoutez l'`StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`Événement.

  Solution : limitez la résolution vidéo à 720p lorsque vous invoquez ou. `getUserMedia` `getDisplayMedia` Plus précisément, vos valeurs `getUserMedia` et celles des `getDisplayMedia` contraintes pour la largeur et la hauteur ne doivent pas dépasser 921600 (1280* 720) lorsqu'elles sont multipliées ensemble.

Limitations de Safari

- Le refus d'une demande d'autorisation nécessite de réinitialiser l'autorisation dans les paramètres du site web Safari au niveau du système d'exploitation.
- Safari ne détecte pas nativement tous les périphériques aussi efficacement que Firefox ou Chrome. Par exemple, la caméra virtuelle OBS n'est pas détectée.

Limitations de Firefox

- Les autorisations système doivent être activées pour que Firefox puisse partager l'écran. Après les avoir activées, l'utilisateur doit redémarrer Firefox pour qu'il fonctionne correctement ; sinon, si les autorisations sont perçues comme bloquées, le navigateur lancera une NotFoundErrorexception.
- La méthode `getCapabilities` est manquante. Cela signifie que les utilisateurs ne peuvent pas obtenir la résolution ou le rapport hauteur/largeur de la piste multimédia. Consultez ce thread Bugzilla.
- Plusieurs propriétés `AudioContext` sont manquantes, par exemple la latence et le nombre de canaux. Cela peut poser un problème aux utilisateurs expérimentés qui souhaitent manipuler les pistes audio.
- Les flux de caméra provenant de `getUserMedia` sont limités au format 4:3 sur macOS. Voir le thread Bugzilla 1 et le thread Bugzilla 2.
- La capture audio n'est pas prise en charge avec `getDisplayMedia`. Consultez ce thread Bugzilla.
- La fréquence d'images lors de la capture d'écran n'est pas optimale (environ 15 images par seconde ?). Consultez ce thread Bugzilla.

Limites du Web mobile

- getDisplayMedialе partage d'écran n'est pas pris en charge sur les appareils mobiles.

  Solution de contournement : aucune.
- Le participant met 15 à 30 secondes pour partir lorsqu'il ferme un navigateur sans appeler `leave()`.

Solution : Ajoutez une interface utilisateur qui encourage les utilisateurs à se déconnecter correctement.

- L’application d’arrière-plan entraîne l’arrêt de la diffusion de vidéos.

  Solution : Affichez une liste d’interfaces utilisateur lorsque le diffuseur de publication est suspendu.

- La fréquence d’images vidéo chute pendant environ 5 secondes après avoir désactivé une caméra sur les appareils Android.

  Solution de contournement : aucune.

- Le flux vidéo est étiré en rotation pour iOS 16.0.

  Solution : Affichez une interface utilisateur décrivant ce problème connu du système d’exploitation.

- Commuter le périphérique d’entrée audio commute automatiquement le périphérique de sortie audio.

  Solution de contournement : aucune.

- En arrière-plan du navigateur, le flux de publication devient noir et ne produit que du son.

  Solution de contournement : aucune. C'est pour des raisons de sécurité.

Gestion des erreurs

Cette section fournit une vue d’ensemble des conditions d’erreur, de la manière dont le kit SDK de diffusion pour le Web les signale à l’application et de ce que l’application doit faire lorsque de telles erreurs se produisent. Il existe quatre catégories d’erreurs :

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

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

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

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

Erreurs d’instanciation d’étape

L’instanciation d’étape ne valide pas les jetons à distance, mais elle permet de détecter certains problèmes de base liés aux jetons qui peuvent être validés côté client. Par conséquent, le kit SDK peut générer une erreur.

Jeton de participant incorrect

Cela se produit lorsque le jeton d’étape est incorrect. Lors de l’instanciation d’une étape, le kit SDK génère une erreur avec le message suivant : « Erreur lors de l’analyse du jeton d’étape ».

Action : créez un jeton valide et réessayez l’instanciation.

Erreurs de jointure d’étapes

Il s’agit des erreurs qui peuvent se produire lors de la première tentative de rejoindre une étape.

L’étape a été supprimée

Cela se produit lorsque vous rejoignez une étape (associée à un jeton) qui a été supprimée. La méthode `join` SDK génère une erreur avec le message suivant : « InitialConnectTimedOut après 10 secondes ».

Action : créez un jeton valide avec une nouvelle étape et réessayez de vous connecter.

Jeton de participant expiré

Cela se produit lorsque le jeton a expiré. La méthode `join` SDK génère une erreur avec le message suivant : « Le jeton a expiré et n’est plus valide ».

Action : créez un nouveau jeton et réessayez de vous connecter.

Jeton de participant non valide ou révoqué

Cela se produit lorsque le jeton n'est pas valide ou a été révoqué/déconnecté. La méthode `join` SDK génère une erreur avec le message suivant : « InitialConnectTimedOut après 10 secondes ».

Action : créez un nouveau jeton et réessayez de vous connecter.

Jeton déconnecté

Cela se produit lorsque le jeton d'étape n'est pas incorrect mais qu'il est rejeté par le serveur Stages. La méthode `join` SDK génère une erreur avec le message suivant : « InitialConnectTimedOut après 10 secondes ».

Action : créez un jeton valide et réessayez de vous connecter.

Erreurs réseau lors de la connexion initiale

Cela se produit lorsque le kit SDK ne parvient pas à contacter le serveur Stages pour établir une connexion. La méthode `join` SDK génère une erreur avec le message suivant : « InitialConnectTimedOut après 10 secondes ».

Action : attendez que la connectivité de l'appareil soit rétablie et réessayez de vous connecter.

Erreurs réseau lorsque vous êtes déjà connecté

En cas de perte de la connexion réseau de l'appareil, le kit SDK risque de perdre sa connexion aux serveurs Stage. Des erreurs peuvent s'afficher dans la console car le kit SDK ne peut plus accéder aux services backend. Les publications sur https://broadcast.stats.live-video.net échoueront.

Si vous publiez et/ou que vous vous abonnez, des erreurs liées à des tentatives de publication/d'abonnement apparaîtront dans la console.

En interne, le kit SDK tentera de se reconnecter avec une stratégie de backoff exponentiel.

Action : attendez que la connectivité de l'appareil soit rétablie. Si vous publiez ou si vous vous abonnez, actualisez la stratégie pour garantir la republication de votre (vos) flux multimédias.

Erreurs de publication et d’abonnement

Erreur de publication : états de publication

Le kit SDK signale `ERRORED` lorsqu’une publication échoue. Cela peut se produire en raison des conditions du réseau ou si une étape est à pleine capacité pour les éditeurs.

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

Action : actualisez la stratégie pour tenter de republier votre (vos) flux multimédias.

Erreurs d’abonnement

Le kit SDK signale `ERRORED` lorsqu’un abonnement échoue. Cela peut se produire en raison des conditions du réseau ou si une étape est à pleine capacité pour les abonnés.

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

Action : actualisez la stratégie pour essayer un nouvel abonnement.

SDK de diffusion IVS : guide pour Android (Streaming en temps réel)

Le SDK de diffusion Android par streaming en temps réel IVS permet aux participants d’envoyer et de recevoir des vidéos sur Android.

Le package `com.amazonaws.ivs.broadcast` implémente l’interface décrite dans ce document. Le SDK prend en charge les opérations suivantes :

- Rejoindre une étape

- Publier du contenu multimédia à l'intention des autres participants de l'étape
- S'abonner à du contenu multimédia d'autres participants de l'étape
- Gérer et surveiller la vidéo et le son publiés sur l'étape
- Obtenir des statistiques WebRTC pour chaque connexion d'appairage
- Toutes les opérations à partir du SDK de diffusion Android par streaming à faible latence

Dernière version du SDK de diffusion Android : 1.14.1 (notes de version)

Documentation de référence : pour plus d'informations sur les méthodes les plus importantes disponibles dans le SDK de diffusion Amazon IVS pour Android, consultez la documentation de référence à l'adresse https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/android/.

Exemple de code : consultez le référentiel d'exemples Android à l'adresse GitHub : https://github.com/aws-samples/ amazon-ivs-broadcast-android -sample.

Exigences de la plateforme : Android 9.0 et versions ultérieures.

Démarrage

Installer la bibliothèque

Pour ajouter la bibliothèque de diffusion Android Amazon IVS à votre environnement de développement Android, ajoutez la bibliothèque au fichier `build.gradle` de votre module, comme indiqué ici (pour la dernière version du kit SDK de diffusion Amazon IVS) :

```
repositories {
    mavenCentral()
}

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

Ajoutez l'autorisation suivante à votre manifeste pour permettre au kit SDK d'activer et de désactiver le haut-parleur :

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

Vous pouvez également installer le kit SDK manuellement, en téléchargeant la dernière version à partir du lien suivant :

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

N'oubliez pas de télécharger le `aar` en ajoutant le suffixe `-stages`.

Demander des autorisations

Votre appli doit demander l'autorisation d'accéder à la caméra et au micro de l'utilisateur. (Ce n'est pas spécifique à Amazon IVS ; cette autorisation est requise pour toute application devant accéder aux caméras et aux micros.)

Ici, nous vérifions si l'utilisateur a déjà accordé des autorisations et, dans le cas contraire, nous les demandons :

```
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;
    }
}
```

Ici, nous obtenons la réponse de l'utilisateur :

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

Publication et abonnement

Concepts

Trois concepts de base sous-tendent la fonctionnalité temps réel : scène, stratégie et moteur de rendu. L'objectif de la conception consiste à minimiser la quantité de logique côté client nécessaire à la création d'un produit fonctionnel.

Étape

La classe `Stage` est le principal point d'interaction entre l'application hôte et le kit SDK. Il représente l'étape elle-même et est utilisé pour rejoindre et quitter l'étape. La création et la participation à une étape nécessitent une chaîne de jetons valide et non expirée provenant du plan de contrôle (représentée par `token`). Il est très facile de rejoindre une étape et de la quitter.

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

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

stage.leave();
```

La classe `Stage` est également l'endroit où vous pouvez joindre le `StageRenderer` :

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

Strategy

L'interface `Stage.Strategy` permet à l'application hôte de communiquer l'état de l'étape souhaité au kit SDK. Trois fonctions doivent être mises en œuvre : `shouldSubscribeToParticipant`, `shouldPublishFromParticipant` et `stageStreamsToPublishForParticipant`. Elles sont toutes abordées ci-dessous.

Abonnement aux participants

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

Lorsqu'un participant distant rejoint l'étape, le kit SDK interroge l'application hôte sur l'état de l'abonnement souhaité pour ce participant. Les options sont `NONE`, `AUDIO_ONLY` et `AUDIO_VIDEO`. Lors d'un renvoi d'une valeur pour cette fonction, l'application hôte n'a pas à se soucier de l'état de publication, de l'état actuel de l'abonnement ou de l'état de la connexion de l'étape. Si `AUDIO_VIDEO` est renvoyé, le kit SDK attend que le participant distant effectue une publication avant de s'abonner, puis il met à jour l'application hôte par le biais du moteur de rendu tout au long du processus.

Voici un exemple d'implémentation :

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

Il s'agit de l'implémentation complète de cette fonction pour une application hôte qui souhaite toujours que tous les participants se voient mutuellement, comme une application de chat vidéo.

Des implémentations plus avancées sont également possibles. Utilisez la propriété `userInfo` sur `ParticipantInfo` pour vous abonner de manière sélective aux participants en fonction des attributs fournis par le serveur :

```
@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;
 }
}
```

Elle peut servir à créer une scène où les modérateurs peuvent surveiller tous les invités sans être vus ou entendus. L'application hôte pourrait utiliser une logique métier supplémentaire pour permettre aux modérateurs de se voir mutuellement tout en restant invisible pour les invités.

Publication

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

Une fois connecté à l'étape, le kit SDK interroge l'application hôte pour savoir si un participant en particulier doit effectuer une publication. Elle n'est invoquée que pour les participants locaux autorisés à publier sur la base du jeton fourni.

Voici un exemple d'implémentation :

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

Il s'agit d'une application de chat vidéo standard dans laquelle les utilisateurs souhaitent toujours publier. Ils peuvent désactiver et réactiver leur son et leur vidéo pour être instantanément masqués ou vus/entendus. (Ils peuvent également utiliser la fonction de publication/d'annulation de la publication, mais c'est beaucoup plus lent. Il est préférable de désactiver ou de réactiver le son dans les cas d'utilisation où il est souvent souhaitable de modifier la visibilité.)

Choix des flux à publier

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

Lors de la publication, cette fonction est utilisée pour déterminer les flux audio et vidéo à publier. Ce point est abordé plus en détail dans la section Publier un flux multimédia.

Mise à jour de la stratégie

La stratégie se veut dynamique : les valeurs renvoyées par n'importe laquelle des fonctions ci-dessus peuvent être modifiées à tout moment. Par exemple, si l'application hôte ne souhaite pas publier

tant que l'utilisateur final n'a pas appuyé sur un bouton, vous pouvez renvoyer une variable depuis `shouldPublishFromParticipant` (quelque chose comme `hasUserTappedPublishButton`). Lorsque cette variable change en fonction d'une interaction de l'utilisateur final, appelez `stage.refreshStrategy()` pour signaler au kit SDK qu'il doit interroger la stratégie pour connaître les dernières valeurs, en appliquant uniquement les éléments qui ont changé. Si le kit SDK constate que la valeur `shouldPublishFromParticipant` a changé, il lance le processus de publication. Si les requêtes du kit SDK et toutes les fonctions renvoient la même valeur qu'auparavant, l'appel `refreshStrategy` n'apportera aucune modification à l'étape.

Si la valeur de retour de `shouldSubscribeToParticipant` passe de `AUDIO_VIDEO` à `AUDIO_ONLY`, le flux vidéo sera supprimé pour tous les participants dont les valeurs renvoyées ont été modifiées, s'il existait déjà un flux vidéo.

En général, l'étape utilise la stratégie pour appliquer au mieux la différence entre les stratégies précédentes et actuelles, sans que l'application hôte n'ait à se soucier de tout l'état requis pour la gérer correctement. Pour cette raison, et pour réduire les frais, envisagez d'appeler `stage.refreshStrategy()`, car cela ne fait rien à moins que la stratégie ne change.

Moteur de rendu

L'interface `StageRenderer` communique l'état de l'étape à l'application hôte. Les mises à jour de l'interface utilisateur de l'application hôte peuvent généralement être entièrement optimisées par les événements fournis par le moteur de rendu. Le moteur de rendu fournit les fonctions suivantes :

```
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);
```

Pour la plupart de ces méthodes, les `Stage` et `ParticipantInfo` correspondantes sont fournies.

Les informations fournies par le moteur de rendu ne devraient pas avoir d'impact sur les valeurs de retour de la stratégie. Par exemple, la valeur de retour de `shouldSubscribeToParticipant` ne devrait pas changer lors de l'appel de on`ParticipantPublishStateChanged`. Si l'application hôte souhaite s'abonner à un participant en particulier, elle doit renvoyer le type d'abonnement souhaité, quel que soit l'état de publication de ce participant. Le kit SDK est chargé de s'assurer que l'état souhaité de la stratégie est appliqué au bon moment en fonction de l'état de l'étape.

Le `StageRenderer` peut être attaché à la classe de l'étape :

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

Notez que seuls les participants à la publication déclenchent on`ParticipantJoined` et que chaque fois qu'un participant arrête de publier ou quitte la session de l'étape, on`ParticipantLeft` est déclenché.

Publier un flux multimédia

Les appareils locaux tels que les microphones et les caméras intégrés sont découverts via `DeviceDiscovery`. Voici un exemple de sélection de la caméra frontale et du microphone par défaut, puis de leur renvoi en tant que `LocalStageStreams` à publier par le SDK :

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

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

Device frontCamera = null;
Device microphone = null;

// Create streams using the front camera, first microphone
for (Device device : devices) {
```

```
 Device.Descriptor descriptor = device.getDescriptor();
 if (!frontCamera && descriptor.type == Device.Descriptor.DeviceType.Camera &&
 descriptor.position = Device.Descriptor.Position.FRONT) {
  front Camera = device;
 }
 if (!microphone && descriptor.type == Device.Descriptor.DeviceType.Microphone) {
  microphone = device;
 }
}

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

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

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

Afficher et supprimer des participants

Une fois l'abonnement effectué, vous recevrez un tableau d'objets `StageStream` via la fonction on`StreamsAdded` du moteur de rendu. Vous pouvez récupérer l'aperçu à partir d'un `ImageStageStream` :

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

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

Vous pouvez récupérer les statistiques au niveau de l'audio à partir d'un `AudioStageStream` :

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

```
});
```

Lorsqu'un participant arrête de publier ou en cas de désabonnement, la fonction `onStreamsRemoved` est appelée avec les flux qui ont été supprimés. Les applications hôte doivent utiliser ce signal pour supprimer le flux vidéo du participant de la hiérarchie des vues.

`onStreamsRemoved` est invoqué pour tous les scénarios dans lesquels un flux peut être supprimé, notamment :

- Le participant distant arrête de publier.
- Un appareil local se désabonne ou modifie l'abonnement de `AUDIO_VIDEO` en `AUDIO_ONLY`.
- Le participant distant quitte l'étape.
- Le participant local quitte l'étape.

Comme `onStreamsRemoved` est invoqué pour tous les scénarios, aucune logique métier personnalisée n'est requise pour supprimer des participants de l'interface utilisateur lors des opérations de départ locales ou à distance.

Désactiver et réactiver le son des flux de médias sociaux

Les objets `LocalStageStream` ont une fonction `setMuted` qui contrôle si le son du flux est désactivé. Cette fonction peut être appelée sur le flux avant ou après son renvoi par la fonction de stratégie `streamsToPublishForParticipant`.

Important : si une nouvelle instance d'objet `LocalStageStream` est renvoyée par `streamsToPublishForParticipant` après un appel à `refreshStrategy`, l'état muet du nouvel objet de flux est appliqué à l'étape. Lorsque vous créez des instances `LocalStageStream`, veillez à ce que l'état muet attendu soit conservé.

Surveiller l'état muet du contenu multimédia des participants distants

Lorsqu'un participant modifie l'état muet de son flux vidéo ou audio, la fonction `onStreamMutedChanged` du moteur de rendu est invoquée avec une liste des flux qui ont changé. Utilisez la méthode `getMuted` sur `StageStream` pour mettre à jour votre interface utilisateur en conséquence.

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

```
 for (StageStream stream : streams) {
  boolean muted = stream.getMuted();
  // handle UI changes
 }
}
```

Obtenir les statistiques WebRTC

Pour obtenir les dernières statistiques WebRTC relatives à un flux de publication ou d'abonnement, utilisez `requestRTCStats` sur `StageStream`. Lorsqu'une collecte est réalisée, vous recevez des statistiques via le `StageStream.Listener` que vous pouvez définir sur `StageStream`.

```
stream.requestRTCStats();

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

Obtenir les attributs des participants

Si vous spécifiez des attributs dans la demande de point de terminaison `CreateParticipantToken`, vous pouvez les voir dans les propriétés `ParticipantInfo` :

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

Poursuivre la session en arrière-plan

Lorsque l'application passe en arrière-plan, vous souhaiterez peut-être vous abonner uniquement au contenu audio des autres participants distants ou arrêter de le publier. Pour ce faire, mettez à

jour votre implémentation `Strategy` pour arrêter la publication et abonnez-vous à `AUDIO_ONLY` (ou `NONE`, le cas échéant).

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

Activer/désactiver le codage en couches avec Simulcast

Lors de la publication d'un flux multimédia, le SDK transmet des flux vidéo de haute et de faible qualité, afin que les participants distants puissent s'abonner au flux même s'ils disposent d'une bande passante descendante limitée. L'encodage en couches avec simulcast est activé par défaut. Vous pouvez le désactiver en utilisant la classe `StageVideoConfiguration.Simulcast` :

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

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

```
// Other Stage implementation code
```

Limitations relatives à la configuration vidéo

Le kit SDK ne permet pas de forcer l'utilisation du mode portrait ou paysage à l'aide de `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)`. En orientation portrait, la plus petite dimension est utilisée comme largeur, tandis qu'en orientation paysage, il s'agit de la hauteur. Cela signifie que les deux appels suivants à `setSize` ont le même effet sur la configuration vidéo :

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

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

Gestion des problèmes de réseau

En cas de perte de la connexion réseau de l'appareil local, le kit SDK essaie de se reconnecter en interne sans aucune action de l'utilisateur. Dans certains cas, le kit SDK échoue et une action de l'utilisateur est requise. Deux erreurs principales sont liées à la perte de la connexion réseau :

- Code d'erreur 1400, message : « PeerConnection est perdu en raison d'une erreur réseau inconnue »
- Code d'erreur 1300, message : « Le nombre de nouvelles tentatives est épuisé »

Si la première erreur est reçue, mais pas la seconde, cela signifie que le kit SDK est toujours connecté à l'étape et qu'il essaiera de rétablir ses connexions automatiquement. Par mesure de sécurité, vous pouvez appeler `refreshStrategy` sans modifier les valeurs de retour de la méthode stratégique, afin de déclencher une tentative de reconnexion manuelle.

Si la deuxième erreur est reçue, les tentatives de reconnexion du kit SDK ont échoué et l'appareil local n'est plus connecté à l'étape. Dans ce cas, essayez de rejoindre l'étape en appelant `join` une fois votre connexion réseau rétablie.

En général, le fait de rencontrer des erreurs après avoir correctement rejoint une étape indique que le kit SDK n'a pas réussi à rétablir la connexion. Créez un objet `Stage` et essayez de le joindre lorsque les conditions du réseau s'améliorent.

Utilisation de microphones Bluetooth

Pour publier à l'aide de microphones Bluetooth, vous devez établir une connexion Bluetooth SCO :

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

Problèmes connus et solutions de contournement

- Lorsqu'un appareil Android se met en veille et en sort, il est possible que l'aperçu soit figé.

  Solution de contournement : créez et utilisez une `Stage`.
- Lorsqu'un participant rejoint la session avec un jeton utilisé par un autre participant, la première connexion est déconnectée sans erreur spécifique.

  Solution de contournement : aucune.
- Dans de rares cas, le diffuseur de publication publie, alors que l'état de la publication reçu par les abonnés est `inactive`.

  Solution de contournement : essayez de quitter la session, puis de la rejoindre. Si le problème persiste, créez un jeton pour le diffuseur de publication.
- Un problème rare de distorsion audio peut survenir par intermittence pendant une session d'étape, généralement lors d'appels de plus longue durée.

  Solution de contournement : le participant dont le son est déformé peut soit quitter la session et la rejoindre, soit annuler la publication et republier son contenu audio pour ainsi corriger le problème.
- Les microphones externes ne sont pas pris en charge lors de la publication sur une étape.

  Solution de contournement : n'utilisez pas de microphone externe connecté via USB pour publier sur une étape.
- La publication sur une étape avec le partage d'écran en utilisant `createSystemCaptureSources` n'est pas prise en charge.

  Solution de contournement : gérez la capture du système manuellement à l'aide de sources d'entrée d'image personnalisées et de sources d'entrée audio personnalisées.

- Lorsqu'une `ImagePreviewView` est supprimée d'un parent (par exemple, `removeView()` est appelée au niveau du parent), la `ImagePreviewView` est immédiatement lancée. La `ImagePreviewView` n'affiche aucun cadre lorsqu'elle est ajoutée à une autre vue parent.

  Solution de contournement : demandez un autre aperçu à l'aide de `getPreview`.

- Lorsque vous rejoignez une étape avec un Samsung Galaxy S22/+ doté d'Android 12, vous pouvez rencontrer une erreur 1401 et l'appareil local ne parvient pas à rejoindre l'étape, ou le fait sans émettre de son.

  Solution de contournement : passez à Android 13.

- Lorsque vous rejoignez une étape avec un Nokia X20 sous Android 13, il se peut que l'appareil photo ne parvienne pas à s'ouvrir et une exception est déclenchée.

  Solution de contournement : aucune.

- Les appareils équipés du chipset MediaTek Helio risquent de ne pas restituer correctement les vidéos des participants distants.

  Solution de contournement : aucune.

- Sur certains appareils, le système d'exploitation de l'appareil peut choisir un microphone différent de celui sélectionné via le kit SDK. Cela est dû au fait que le kit SDK de diffusion Amazon IVS ne peut pas contrôler la façon dont la route audio `VOICE_COMMUNICATION` est définie, car elle varie en fonction des fabricants d'appareils.

  Solution de contournement : aucune.

- Certains encodeurs vidéo Android ne peuvent pas être configurés avec une taille vidéo inférieure à 176 x 176. La configuration d'une taille plus petite provoque une erreur et empêche le streaming.

  Solution : ne configurez pas la taille de la vidéo pour qu'elle soit inférieure à 176 x 176.

Gestion des erreurs

Erreurs fatales ou non fatales

L'objet d'erreur possède un champ booléen « est fatal » de `BroadcastException`.

En général, les erreurs fatales sont liées à la connexion au serveur Stages (soit la connexion ne peut pas être établie, soit elle est perdue et ne peut pas être rétablie). L'application doit recréer la scène

et la rejoindre, éventuellement avec un nouveau jeton ou lorsque la connectivité de l'appareil sera rétablie.

Les erreurs non fatales sont généralement liées à l'état de publication/d'abonnement et sont gérées par le kit SDK, qui relance l'opération de publication/abonnement.

Vous pouvez vérifier cette propriété :

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

Erreurs de jonction

Jeton incorrect

Cela se produit lorsque le jeton d'étape est incorrect.

Le kit SDK génère une exception Java à partir d'un appel à `stage.join`, avec le code d'erreur = 1000 et fatal = vrai.

Action : créez un jeton valide et réessayez de vous connecter.

Jeton expiré

Cela se produit lorsque le jeton d'étape est expiré.

Le kit SDK génère une exception Java à partir d'un appel à `stage.join`, avec le code d'erreur = 1001 et fatal = vrai.

Action : créez un nouveau jeton et réessayez de vous connecter.

Jeton non valide ou révoqué

Cela se produit lorsque le jeton d'étape n'est pas incorrect mais qu'il est rejeté par le serveur Stages. Cela est signalé de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le kit SDK appelle `onConnectionStateChanged` avec une exception, avec le code d'erreur = 1026 et fatal = vrai.

Action : créez un jeton valide et réessayez de vous connecter.

Erreurs réseau lors de la connexion initiale

Cela se produit lorsque le kit SDK ne parvient pas à contacter le serveur Stages pour établir une connexion. Cela est signalé de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le kit SDK appelle `onConnectionStateChanged` avec une exception, avec le code d'erreur = 1300 et fatal = vrai.

Action : attendez que la connectivité de l'appareil soit rétablie et réessayez de vous connecter.

Erreurs réseau lorsque vous êtes déjà connecté

En cas de perte de la connexion réseau de l'appareil, le kit SDK risque de perdre sa connexion aux serveurs Stage. Cela est signalé de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le kit SDK appelle `onConnectionStateChanged` avec une exception, avec le code d'erreur = 1300 et fatal = vrai.

Action : attendez que la connectivité de l'appareil soit rétablie et réessayez de vous connecter.

Erreurs de publication/d'abonnement

Initial

Il existe plusieurs erreurs :

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

Ceux-ci sont signalés de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le kit SDK relance l'opération un nombre limité de fois. Lors des nouvelles tentatives, l'état de publication/d'abonnement est `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Si les nouvelles tentatives réussissent, l'état passe à `PUBLISHED` / `SUBSCRIBED`.

Le kit SDK appelle `onError` avec le code d'erreur approprié et fatal = faux.

Action : aucune action n'est nécessaire, car le kit SDK réessaie automatiquement. Le cas échéant, l'application peut actualiser la stratégie pour forcer d'autres tentatives.

Déjà établi, puis échec

Une publication ou un abonnement peut échouer une fois qu'il a été établi, probablement en raison d'une erreur réseau. Le code d'erreur pour une « connexion d'appairage perdue en raison d'une erreur réseau » est 1400.

Cela est signalé de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le kit SDK relance l'opération de publication/d'abonnement. Lors des nouvelles tentatives, l'état de publication/d'abonnement est `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Si les nouvelles tentatives réussissent, l'état passe à `PUBLISHED` / `SUBSCRIBED`.

Le kit SDK appelle `onError` avec le code d'erreur = 1400 et fatal = false.

Action : aucune action n'est nécessaire, car le kit SDK réessaie automatiquement. Le cas échéant, l'application peut actualiser la stratégie pour forcer d'autres tentatives. En cas de perte totale de connectivité, il est probable que la connexion à Stages échoue également.

SDK de diffusion IVS : guide pour iOS (Streaming en temps réel)

Le SDK de diffusion iOS par streaming en temps réel IVS permet aux participants d'envoyer et de recevoir des vidéos sur iOS.

Le module `AmazonIVSBroadcast` implémente l'interface décrite dans ce document. Les opérations suivantes sont prises en charge :

- Rejoindre une étape
- Publier du contenu multimédia à l'intention des autres participants de l'étape
- S'abonner à du contenu multimédia d'autres participants de l'étape
- Gérer et surveiller la vidéo et le son publiés sur l'étape

- Obtenir des statistiques WebRTC pour chaque connexion d’appairage
- Toutes les opérations à partir du SDK de diffusion iOS par streaming à faible latence

Dernière version du SDK de diffusion iOS : 1.14.1 (notes de version)

Documentation de référence : pour plus d'informations sur les méthodes les plus importantes disponibles dans le SDK de diffusion Amazon IVS pour iOS, consultez la documentation de référence à l'adresse https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios/.

Exemple de code : consultez le référentiel d'exemples iOS à l'adresse GitHub : https://github.com/ aws-samples/ amazon-ivs-broadcast-ios -sample.

Exigences de la plateforme : iOS 14 ou version ultérieure

Démarrage

Installer la bibliothèque

Nous vous recommandons d'intégrer le SDK de diffusion via CocoaPods. (Vous pouvez également ajouter manuellement le cadre à votre projet.)

Recommandé : intégrer le SDK de diffusion () CocoaPods

La fonctionnalité temps réel est publiée en tant que sous-spécification du SDK de diffusion iOS par streaming à faible latence. Les clients peuvent ainsi choisir de l’inclure ou de l’exclure en fonction de leurs besoins en fonctionnalités. L’inclusion augmente la taille de l’emballage.

Les communiqués sont publiés CocoaPods sous le nom`AmazonIVSBroadcast`. Ajoutez cette dépendance à votre Podfile :

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

Exécutez `pod install` et le kit SDK sera disponible dans votre `.xcworkspace`.

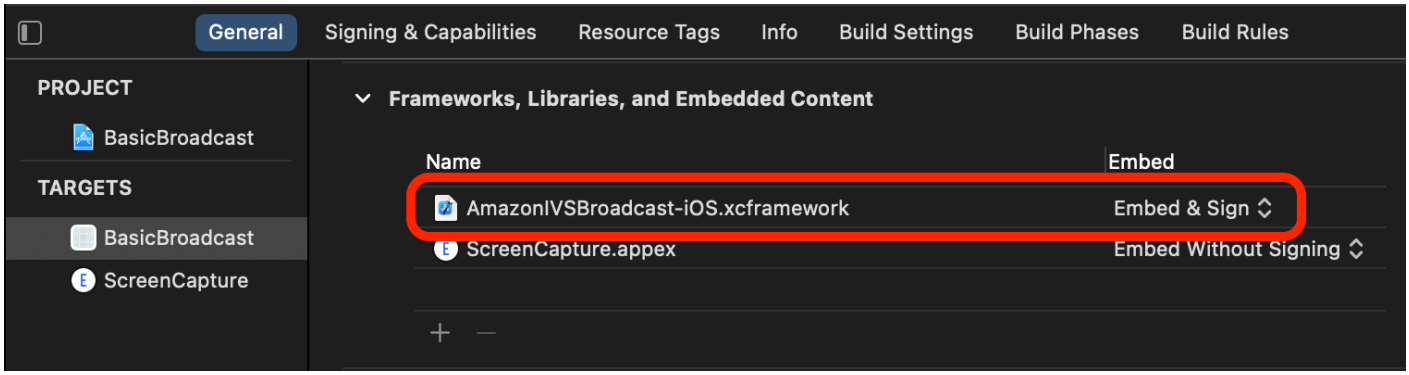
Important : le SDK de diffusion par streaming en temps réel IVS (c’est-à-dire avec la sous-spécification de scène) inclut toutes les fonctionnalités du SDK de diffusion par streaming à faible latence. Il n’est pas possible d’intégrer les deux kits SDK dans le même projet. Si vous ajoutez la sous-spécification de stage via CocoaPods à votre projet, veillez à supprimer toutes les autres lignes

du Podfile qui le contient. `AmazonIVSBroadcast` Par exemple, vous n'avez pas ces deux lignes dans votre Podfile :

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

Autre approche : installer manuellement le cadre

1. Téléchargez la dernière version [sur https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip](https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip).
2. Extrayez le contenu de l'archive. `AmazonIVSBroadcast.xcframework` contient le kit SDK pour l'appareil et le simulateur.
3. Intégrez `AmazonIVSBroadcast.xcframework` en le faisant glisser dans la section Frameworks, Libraries, and Embedded Content (Cadre, bibliothèques et contenu intégré) de l'onglet General (Général) de votre cible d'application.



Demander des autorisations

Votre appli doit demander l'autorisation d'accéder à la caméra et au micro de l'utilisateur. (Ce n'est pas spécifique à Amazon IVS ; cette autorisation est requise pour toute application devant accéder aux caméras et aux micros.)

Ici, nous vérifions si l'utilisateur a déjà accordé des autorisations et, dans le cas contraire, nous les demandons :

```
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.
}
```

Vous devez demander des autorisations pour les types de médias `.video` et `.audio` si vous souhaitez accéder aux caméras et aux microphones, respectivement.

Vous devez également ajouter des entrées pour `NSCameraUsageDescription` et `NSMicrophoneUsageDescription` à votre `Info.plist`. Sinon, votre application se bloquera lorsque vous essayerez de demander des autorisations.

Désactiver le minuteur d'inactivité de l'application

Cette action est facultative, mais recommandée. Elle empêche votre appareil de se mettre en veille lors de l'utilisation du kit SDK de diffusion, ce qui pourrait interrompre la diffusion.

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

Publication et abonnement

Concepts

Trois concepts de base sous-tendent la fonctionnalité temps réel : scène, stratégie et moteur de rendu. L'objectif de la conception consiste à minimiser la quantité de logique côté client nécessaire à la création d'un produit fonctionnel.

Étape

La classe `IVSStage` est le principal point d'interaction entre l'application hôte et le kit SDK. La classe représente l'étape elle-même et est utilisée pour rejoindre et quitter l'étape. La création ou la participation à une étape nécessitent une chaîne de jetons valide et non expirée provenant du plan de contrôle (représentée par `token`). Il est très facile de rejoindre une étape et de la quitter.

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

try stage.join()

stage.leave()
```

La classe `IVSStage` est également l'endroit où vous pouvez joindre le `IVSStageRenderer` et le `IVSErrorDelegate` :

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

Strategy

Le protocole `IVSStageStrategy` permet à l'application hôte de communiquer l'état de l'étape souhaité au kit SDK. Trois fonctions doivent être mises en œuvre :`shouldSubscribeToParticipant`, `shouldPublishParticipant` et `streamsToPublishForParticipant`. Elles sont toutes abordées ci-dessous.

Abonnement aux participants

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

Lorsqu'un participant distant rejoint une étape, le kit SDK interroge l'application hôte sur l'état de l'abonnement souhaité pour ce participant. Les options sont `.none`, `.audioOnly` et `.audioVideo`. Lors d'un renvoi d'une valeur pour cette fonction, l'application hôte n'a pas à se soucier de l'état de publication, de l'état actuel de l'abonnement ou de l'état de la connexion de l'étape. Si `.audioVideo` est renvoyé, le kit SDK attend que le participant distant effectue une publication avant de s'abonner, puis il met à jour l'application hôte par le biais du moteur de rendu tout au long du processus.

Voici un exemple d'implémentation :

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

Il s'agit de l'implémentation complète de cette fonction pour une application hôte qui souhaite toujours que tous les participants se voient mutuellement, comme une application de chat vidéo.

Des implémentations plus avancées sont également possibles. Utilisez la propriété `attributes` sur `IVSParticipantInfo` pour vous abonner de manière sélective aux participants en fonction des attributs fournis par le serveur :

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

Elle peut servir à créer une scène où les modérateurs peuvent surveiller tous les invités sans être vus ou entendus. L'application hôte pourrait utiliser une logique métier supplémentaire pour permettre aux modérateurs de se voir mutuellement tout en restant invisible pour les invités.

Publication

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

Une fois connecté à l'étape, le kit SDK interroge l'application hôte pour savoir si un participant en particulier doit effectuer une publication. Elle n'est invoquée que pour les participants locaux autorisés à publier sur la base du jeton fourni.

Voici un exemple d'implémentation :

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

Il s'agit d'une application de chat vidéo standard dans laquelle les utilisateurs souhaitent toujours publier. Ils peuvent désactiver et réactiver leur son et leur vidéo pour être instantanément masqués ou vus/entendus. (Ils peuvent également utiliser la fonction de publication/d'annulation de la publication, mais c'est beaucoup plus lent. Il est préférable de désactiver ou de réactiver le son dans les cas d'utilisation où il est souvent souhaitable de modifier la visibilité.)

Choix des flux à publier

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

Lors de la publication, cette fonction est utilisée pour déterminer les flux audio et vidéo à publier. Ce point est abordé plus en détail dans la section Publier un flux multimédia.

Mise à jour de la stratégie

La stratégie se veut dynamique : les valeurs renvoyées par n'importe laquelle des fonctions ci-dessus peuvent être modifiées à tout moment. Par exemple, si l'application hôte ne souhaite pas publier tant que l'utilisateur final n'a pas appuyé sur un bouton, vous pouvez renvoyer une variable depuis `shouldPublishParticipant` (quelque chose comme `hasUserTappedPublishButton`). Lorsque cette variable change en fonction d'une interaction de l'utilisateur final, appelez `stage.refreshStrategy()` pour signaler au kit SDK qu'il doit interroger la stratégie pour connaître les dernières valeurs, en appliquant uniquement les éléments qui ont changé. Si le kit SDK constate que la valeur `shouldPublishParticipant` a changé, il lance le processus de publication. Si les requêtes du kit SDK et toutes les fonctions renvoient la même valeur qu'auparavant, l'appel `refreshStrategy` n'apportera aucune modification à l'étape.

Si la valeur de retour de `shouldSubscribeToParticipant` passe de `.audioVideo` à `.audioOnly`, le flux vidéo sera supprimé pour tous les participants dont les valeurs renvoyées ont été modifiées, s'il existait déjà un flux vidéo.

En général, l'étape utilise la stratégie pour appliquer au mieux la différence entre les stratégies précédentes et actuelles, sans que l'application hôte n'ait à se soucier de tout l'état requis pour la gérer correctement. Pour cette raison, et pour réduire les frais, envisagez d'appeler `stage.refreshStrategy()`, car cela ne fait rien à moins que la stratégie ne change.

Moteur de rendu

Le protocole `IVSStageRenderer` communique l'état de l'étape à l'application hôte. Les mises à jour de l'interface utilisateur de l'application hôte peuvent généralement être entièrement optimisées par les événements fournis par le moteur de rendu. Le moteur de rendu fournit les fonctions suivantes :

```
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?)
```

Les informations fournies par le moteur de rendu ne devraient pas avoir d'impact sur les valeurs de retour de la stratégie. Par exemple, la valeur de retour de `shouldSubscribeToParticipant` ne devrait pas changer lors de l'appel de `participant:didChangePublishState`. Si l'application hôte souhaite s'abonner à un participant en particulier, elle doit renvoyer le type d'abonnement souhaité, quel que soit l'état de publication de ce participant. Le kit SDK est chargé de s'assurer que l'état souhaité de la stratégie est appliqué au bon moment en fonction de l'état de l'étape.

Notez que seuls les participants à la publication déclenchent `participantDidJoin` et que chaque fois qu'un participant arrête de publier ou quitte la session de l'étape, `participantDidLeave` est déclenché.

Publier un flux multimédia

Les appareils locaux tels que les microphones et les caméras intégrés sont découverts via `IVSDeviceDiscovery`. Voici un exemple de sélection de la caméra frontale et du microphone par défaut, puis de leur renvoi en tant que `IVSLocalStageStreams` à publier par le SDK :

```
let devices = IVSDeviceDiscovery.listLocalDevices()

// Find the camera virtual device, choose the front source, and create a stream
let camera = devices.compactMap({ $0 as? IVSCamera }).first!
let frontSource = camera.listAvailableInputSources().first(where: { $0.position
 == .front })!
camera.setPreferredInputSource(frontSource)
```

```
let cameraStream = IVSLocalStageStream(device: camera)

// Find the microphone virtual device, enable echo cancellation, and create a stream
let microphone = devices.compactMap({ $0 as? IVSMicrophone }).first!
microphone.isEchoCancellationEnabled = true
let microphoneStream = IVSLocalStageStream(device: microphone)

// This is a function on IVSStageStrategy
func stage(_ stage: IVSStage, streamsToPublishForParticipant participant:
 IVSParticipantInfo) -> [IVSLocalStageStream] {
    return [cameraStream, microphoneStream]
}
```

Afficher et supprimer des participants

Une fois l'abonnement effectué, vous recevrez un tableau d'objets `IVSStageStream` via la fonction `didAddStreams` du moteur de rendu. Pour avoir un aperçu ou recevoir des statistiques de niveau audio concernant ce participant, vous pouvez accéder à l'objet `IVSDevice` sous-jacent depuis le flux :

```
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 */
    })
}
```

Lorsqu'un participant arrête de publier ou en cas de désabonnement, la fonction `didRemoveStreams` est appelée avec les flux qui ont été supprimés. Les applications hôte doivent utiliser ce signal pour supprimer le flux vidéo du participant de la hiérarchie des vues.

`didRemoveStreams` est invoqué pour tous les scénarios dans lesquels un flux peut être supprimé, notamment :

- Le participant distant arrête de publier.
- Un appareil local se désabonne ou modifie l'abonnement de `.audioVideo` en `.audioOnly`.
- Le participant distant quitte l'étape.
- Le participant local quitte l'étape.

Comme `didRemoveStreams` est invoqué pour tous les scénarios, aucune logique métier personnalisée n'est requise pour supprimer des participants de l'interface utilisateur lors des opérations de départ locales ou à distance.

Désactiver et réactiver le son des flux de médias sociaux

Les objets `IVSLocalStageStream` ont une fonction `setMuted` qui contrôle si le son du flux est désactivé. Cette fonction peut être appelée sur le flux avant ou après son renvoi par la fonction de stratégie `streamsToPublishForParticipant`.

Important : si une nouvelle instance d'objet `IVSLocalStageStream` est renvoyée par `streamsToPublishForParticipant` après un appel à `refreshStrategy`, l'état muet du nouvel objet de flux est appliqué à l'étape. Lorsque vous créez des instances `IVSLocalStageStream`, veillez à ce que l'état muet attendu soit conservé.

Surveiller l'état muet du contenu multimédia des participants distants

Lorsqu'un participant modifie l'état muet de son flux vidéo ou audio, la fonction `didChangeMutedStreams` du moteur de rendu est invoquée avec un tableau des flux qui ont changé. Utilisez la propriété `isMuted` sur `IVSStageStream` pour mettre à jour votre interface utilisateur en conséquence :

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

Création d'une configuration d'étape

Pour personnaliser les valeurs de la configuration vidéo d'une étape, utilisez `IVSLocalStageStreamVideoConfiguration` :

```
let config = IVSLocalStageStreamVideoConfiguration()
try config.setMaxBitrate(900_000)
try config.setMinBitrate(100_000)
try config.setTargetFramerate(30)
try config.setSize(CGSize(width: 360, height: 640))
config.degradationPreference = .balanced
```

Obtenir les statistiques WebRTC

Pour obtenir les dernières statistiques WebRTC relatives à un flux de publication ou d'abonnement, utilisez `requestRTCStats` sur `IVSStageStream`. Lorsqu'une collecte est réalisée, vous recevez des statistiques via le `IVSStageStreamDelegate` que vous pouvez définir sur `IVSStageStream`. Pour collecter en permanence des statistiques WebRTC, appelez cette fonction sur un `Timer`.

```
func stream(_ stream: IVSStageStream, didGenerateRTCStats stats: [String : [String :
 String]]) {
    for stat in stats {
      for member in stat.value {
         print("stat \(stat.key) has member \(member.key) with value \(member.value)")
      }
   }
}
```

Obtenir les attributs des participants

Si vous spécifiez des attributs dans la demande de point de terminaison `CreateParticipantToken`, vous pouvez les voir dans les propriétés `IVSParticipantInfo` :

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

Poursuivre la session en arrière-plan

Lorsque l'application passe en arrière-plan, vous pouvez rester dans l'étape tout en écoutant le son distant, mais il n'est pas possible de continuer à envoyer vos propres images et sons. Vous devrez mettre à jour votre implémentation `IVSStrategy` pour arrêter la publication et vous abonner à `.audioOnly` (ou `.none`, le cas échéant) :

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

```
    return .audioOnly
}
```

Passez ensuite un appel à `stage.refreshStrategy()`.

Activer/désactiver le codage en couches avec Simulcast

Lors de la publication d'un flux multimédia, le SDK transmet des flux vidéo de haute et de faible qualité, afin que les participants distants puissent s'abonner au flux même s'ils disposent d'une bande passante descendante limitée. L'encodage en couches avec simulcast est activé par défaut. Vous pouvez le désactiver avec `IVSSimulcastConfiguration` :

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

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

// Other Stage implementation code
```

Diffuser la scène sur un canal IVS

Pour diffuser une étape, créez une `IVSBroadcastSession` distincte, puis suivez les instructions habituelles de diffusion avec le kit SDK, décrites ci-dessus. La propriété `device` sur `IVSStageStream` sera soit un `IVSImageDevice` soit un `IVSAudioDevice` comme indiqué dans l'extrait ci-dessus. Ils peuvent être connectés au `IVSBroadcastSession.mixer` pour diffuser l'intégralité de l'étape dans une disposition personnalisable.

Vous pouvez éventuellement composer une scène et la diffuser sur un canal IVS à faible latence, afin de toucher un public plus large. Consultez la section Activation d'hôtes multiples sur un flux Amazon IVS dans le guide de l'utilisateur du streaming à faible latence IVS.

Comment iOS choisit la résolution de la caméra et la fréquence d'images

La caméra gérée par le SDK de diffusion optimise sa résolution et sa fréquence d'images (frames-per-secondou FPS) afin de minimiser la production de chaleur et la consommation d'énergie. Cette section explique comment la résolution et la fréquence d'images sont sélectionnées pour aider les applications hôtes à optimiser leur capacité en fonction de leurs cas d'utilisation.

Lorsque vous créez une `IVSLocalStageStream` avec une `IVSCamera`, la caméra est optimisée pour une fréquence d'images

de `IVSLocalStageStreamVideoConfiguration.targetFramerate` et une résolution de `IVSLocalStageStreamVideoConfiguration.size`. L'appel de `IVSLocalStageStream.setConfiguration` met à jour la caméra avec de nouvelles valeurs.

Aperçu de la caméra

Si vous créez un aperçu d'une `IVSCamera` sans le joindre à une `IVSBroadcastSession` ou une `IVSStage`, la résolution par défaut est de 1080p et la fréquence d'images de 60 i/s.

Diffusion d'une étape

Lorsque vous utilisez une `IVSBroadcastSession` pour diffuser une `IVSStage`, le kit SDK essaie d'optimiser la caméra avec une résolution et une fréquence d'images qui répondent aux critères des deux sessions.

Par exemple, si la configuration de diffusion est définie pour avoir une fréquence d'images de 15 i/s et une résolution de 1080p, alors que la scène a une fréquence d'images de 30 i/s et une résolution de 720p, le kit SDK sélectionnera une configuration de caméra avec une fréquence d'images de 30 i/s et une résolution de 1080p. La `IVSBroadcastSession` supprimera une image sur deux de la caméra et `IVSStage` redimensionnera l'image de 1080p à 720p.

Si une application hôte prévoit d'utiliser `IVSBroadcastSession` et `IVSStage` conjointement avec une caméra, nous recommandons que les propriétés `targetFramerate` et `size` des configurations respectives correspondent. En cas de non-concordance, la caméra pourrait se reconfigurer pendant la capture vidéo, ce qui retarderait brièvement la livraison des échantillons vidéo.

Si le fait d'avoir des valeurs identiques ne correspond pas au scénario d'utilisation de l'application hôte, la création préalable de la caméra de meilleure qualité empêchera la caméra de se reconfigurer elle-même lors de l'ajout de la session de moindre qualité. Par exemple, si vous diffusez en 1080p et 30 i/s, puis que vous rejoignez ultérieurement une scène réglée sur 720p et 30 i/s, la caméra ne se reconfigurera pas d'elle-même et la vidéo se poursuivra sans interruption. Cela est dû au fait que 720p est inférieur ou égal à 1080p et 30 i/s sont inférieures ou égales à 30 i/s.

Fréquences d'images, résolutions et rapports hauteur/largeur arbitraires

La plupart des appareils photo peuvent correspondre exactement aux formats courants, tels que 720p à 30 i/s ou 1080p à 60 i/s. Toutefois, il n'est pas possible de faire correspondre exactement tous les formats. Le kit SDK de diffusion choisit la configuration de la caméra en fonction des règles suivantes (par ordre de priorité) :

1. La largeur et la hauteur de la résolution sont supérieures ou égales à la résolution souhaitée, mais dans le cadre de cette contrainte, la largeur et la hauteur sont aussi petites que possible.
2. La fréquence d'images est supérieure ou égale à la fréquence d'images souhaitée, mais dans les limites de cette contrainte, la fréquence d'images est aussi faible que possible.
3. Le rapport hauteur/largeur correspond au rapport hauteur/largeur souhaité.
4. S'il existe plusieurs formats correspondants, le format présentant le plus grand champ de vision est utilisé.

Voici deux exemples :

- L'application hôte essaie de diffuser en 4K à 120 i/s. La caméra sélectionnée prend uniquement en charge la résolution 4K à 60 i/s ou la résolution 1080p à 120 i/s. Le format sélectionné sera 4K à 60 i/s, car la règle de résolution est plus prioritaire que la règle de fréquence d'images.
- Une résolution irrégulière est demandée, 1910x1070. La caméra utilisera la résolution 1920 x 1080. Attention : si vous choisissez une résolution telle que 1921 x 1080, l'appareil photo passera à la résolution suivante disponible (par exemple, 2592 x 1944), ce qui entraîne une réduction de la bande passante du processeur et de la mémoire.

Et pour Android ?

Android n'ajuste pas sa résolution ou sa fréquence d'images à la volée comme le fait iOS. Cela n'a donc aucune incidence sur le kit SDK de diffusion Android.

Problèmes connus et solutions de contournement

- La modification des itinéraires audio Bluetooth peut être imprévisible. Si vous connectez un nouvel appareil au milieu d'une session, iOS peut ou non modifier automatiquement l'itinéraire d'entrée. En outre, il n'est pas possible de choisir entre plusieurs casques Bluetooth connectés en même temps. Cela se produit à la fois lors de sessions de diffusion et d'étape régulières.

  Solution de contournement : si vous prévoyez d'utiliser un casque Bluetooth, connectez-le avant de démarrer la diffusion ou l'étape et laissez-le connecté tout au long de la session.
- Les participants utilisant un iPhone 14, un iPhone 14 Plus, un iPhone 14 Pro ou un iPhone 14 Pro Max peuvent provoquer un problème d'écho audio pour les autres participants.

  Solution de contournement : les participants utilisant les appareils concernés peuvent utiliser des écouteurs pour éviter que les autres participants ne soient confrontés à un problème d'écho.

- Lorsqu'un participant rejoint la session avec un jeton utilisé par un autre participant, la première connexion est déconnectée sans erreur spécifique.

  Solution de contournement : aucune.

- Dans de rares cas, le diffuseur de publication publie, alors que l'état de la publication reçu par les abonnés est `inactive`.

  Solution de contournement : essayez de quitter la session, puis de la rejoindre. Si le problème persiste, créez un jeton pour le diffuseur de publication.

- Lorsqu'un participant publie ou s'abonne, il est possible de recevoir une erreur avec le code 1400 qui indique une déconnexion due à un problème de réseau, même lorsque le réseau est stable.

  Solution de contournement : essayez de republier ou de vous réabonner.

- Un problème rare de distorsion audio peut survenir par intermittence pendant une session d'étape, généralement lors d'appels de plus longue durée.

  Solution de contournement : le participant dont le son est déformé peut soit quitter la session et la rejoindre, soit annuler la publication et republier son contenu audio pour ainsi corriger le problème.

Gestion des erreurs

Erreurs fatales ou non fatales

L'objet d'erreur possède un champ booléen « est fatal » . Il s'agit d'une entrée de dictionnaire sous `IVSBroadcastErrorIsFatalKey` laquelle contient un booléen.

En général, les erreurs fatales sont liées à la connexion au serveur Stages (soit la connexion ne peut pas être établie, soit elle est perdue et ne peut pas être rétablie). L'application doit recréer la scène et la rejoindre, éventuellement avec un nouveau jeton ou lorsque la connectivité de l'appareil sera rétablie.

Les erreurs non fatales sont généralement liées à l'état de publication/d'abonnement et sont gérées par le kit SDK, qui relance l'opération de publication/abonnement.

Vous pouvez vérifier cette propriété :

```
let nsError = error as NSError
if nsError.userInfo[IVSBroadcastErrorIsFatalKey] as? Bool == true {
```

```
  // the error is fatal
}
```

Erreurs de jonction

Jeton incorrect

Cela se produit lorsque le jeton d'étape est incorrect.

Le SDK génère une exception Swift avec le code d'erreur = 1000 et IVS BroadcastErrorIsFatalKey = YES.

Action : créez un jeton valide et réessayez de vous connecter.

Jeton expiré

Cela se produit lorsque le jeton d'étape est expiré.

Le SDK génère une exception Swift avec le code d'erreur = 1001 et IVS BroadcastErrorIsFatalKey = YES.

Action : créez un nouveau jeton et réessayez de vous connecter.

Jeton non valide ou révoqué

Cela se produit lorsque le jeton d'étape n'est pas incorrect mais qu'il est rejeté par le serveur Stages. Cela est signalé de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le SDK appelle `stage(didChange connectionState, withError error)` avec le code d'erreur = 1026 et IVS BroadcastErrorIsFatalKey = YES.

Action : créez un jeton valide et réessayez de vous connecter.

Erreurs réseau lors de la connexion initiale

Cela se produit lorsque le kit SDK ne parvient pas à contacter le serveur Stages pour établir une connexion. Cela est signalé de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le SDK appelle `stage(didChange connectionState, withError error)` avec le code d'erreur = 1300 et IVS BroadcastErrorIsFatalKey = YES.

Action : attendez que la connectivité de l’appareil soit rétablie et réessayez de vous connecter.

Erreurs réseau lorsque vous êtes déjà connecté

En cas de perte de la connexion réseau de l’appareil, le kit SDK risque de perdre sa connexion aux serveurs Stage. Cela est signalé de façon asynchrone via le moteur de rendu d’étape fourni par l’application.

Le SDK appelle `stage(didChange connectionState, withError error)` avec le code d'erreur = 1300 et la BroadcastErrorIsFatalKey valeur IVS = OUI.

Action : attendez que la connectivité de l’appareil soit rétablie et réessayez de vous connecter.

Erreurs de publication/d’abonnement

Initial

Il existe plusieurs erreurs :

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

Ceux-ci sont signalés de façon asynchrone via le moteur de rendu d’étape fourni par l’application.

Le kit SDK relance l’opération un nombre limité de fois. Lors des nouvelles tentatives, l’état de publication/d’abonnement est `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Si les nouvelles tentatives réussissent, l’état passe à `PUBLISHED` / `SUBSCRIBED`.

Le SDK appelle `IVSErrorSourceDelegate:didEmitError` avec le code d'erreur correspondant et IVS BroadcastErrorIsFatalKey = NO.

Action : aucune action n’est nécessaire, car le kit SDK réessaie automatiquement. Le cas échéant, l’application peut actualiser la stratégie pour forcer d’autres tentatives.

Déjà établi, puis échec

Une publication ou un abonnement peut échouer une fois qu'il a été établi, probablement en raison d'une erreur réseau. Le code d'erreur pour une « connexion d'appairage perdue en raison d'une erreur réseau » est 1400.

Cela est signalé de façon asynchrone via le moteur de rendu d'étape fourni par l'application.

Le kit SDK relance l'opération de publication/d'abonnement. Lors des nouvelles tentatives, l'état de publication/d'abonnement est `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Si les nouvelles tentatives réussissent, l'état passe à `PUBLISHED` / `SUBSCRIBED`.

Le SDK appelle `didEmitError` avec le code d'erreur = 1400 et IVS BroadcastErrorIsFatalKey = NO.

Action : aucune action n'est nécessaire, car le kit SDK réessaie automatiquement. Le cas échéant, l'application peut actualiser la stratégie pour forcer d'autres tentatives. En cas de perte totale de connectivité, il est probable que la connexion à Stages échoue également.

Kit SDK de diffusion IVS : sources d'images personnalisées (Real-Time Streaming)

Les sources d'entrée d'image personnalisées permettent à une application de fournir sa propre entrée d'image au kit SDK de diffusion, au lieu de se limiter aux caméras prédéfinies. Une source d'image personnalisée peut être aussi simple qu'un filigrane semi-transparent ou une scène statique « Je reviens tout de suite », ou elle peut permettre à l'application d'effectuer un traitement personnalisé supplémentaire, comme l'ajout de filtres de beauté à la caméra.

Lorsque vous utilisez une source d'entrée d'image personnalisée pour un contrôle personnalisé de la caméra (par exemple, l'utilisation de bibliothèques de filtres de beauté nécessitant un accès à la caméra), le kit SDK de diffusion n'est plus responsable de la gestion de la caméra. Au lieu de cela, l'application est chargée de gérer correctement le cycle de vie de la caméra. Consultez la documentation officielle de la plateforme sur la façon dont votre application doit gérer la caméra.

Android

Après avoir créé une session `DeviceDiscovery`, créez une source d'entrée d'image :

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

Cette méthode renvoie un `CustomImageSource`, qui est une source basée sur un Android Surface standard. La sous-classe `SurfaceSource` permet le redimensionnement et la rotation. Vous pouvez également créer un `ImagePreviewView` pour afficher un aperçu de son contenu.

Pour récupérer la `Surface` sous-jacente :

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

Cette `Surface` peut être utilisée comme tampon de sortie pour les producteurs d'images tels que Camera2, OpenGL ES et d'autres bibliothèques. Le cas d'utilisation le plus simple consiste à dessiner directement un bitmap statique ou une couleur dans le canevas de la surface. Cependant, de nombreuses bibliothèques (telles que les bibliothèques de filtres de beauté) fournissent une méthode qui permet à une application de spécifier une `Surface` externe pour le rendu. Vous pouvez utiliser une telle méthode pour transmettre cette `Surface` à la bibliothèque de filtres, ce qui permet à cette dernière de produire des images traitées pour la séance de diffusion.

Cette `CustomImageSource` peut être encapsulée dans un `LocalStageStream` et renvoyée par la `StageStrategy` pour être publiée vers une `Stage`.

iOS

Après avoir créé une session `DeviceDiscovery`, créez une source d'entrée d'image :

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

Cette méthode renvoie une `IVSCustomImageSource`, une source d'image qui permet à l'application de soumettre `CMSampleBuffers` manuellement. Pour connaître les formats de pixels pris en charge, consultez la référence du kit SDK de diffusion iOS. Un lien vers la version la plus récente se trouve dans les notes de mise à jour Amazon IVS pour la dernière version du kit SDK de diffusion.

Les exemples soumis à la source personnalisée seront diffusés dans la scène :

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

Pour diffuser des vidéos en streaming, utilisez cette méthode dans un rappel. Par exemple, si vous utilisez la caméra, chaque fois qu'un nouvel exemple de tampon est reçu d'une `AVCaptureSession`, l'application peut transférer l'exemple de tampon vers la source d'image personnalisée. Si vous le souhaitez, l'application peut appliquer un traitement supplémentaire (comme un filtre de beauté) avant de soumettre l'exemple à la source d'image personnalisée.

La `IVSCustomImageSource` peut être encapsulée dans un `IVSLocalStageStream` et renvoyé par la `IVSStageStrategy` pour être publiée vers une `Stage`.

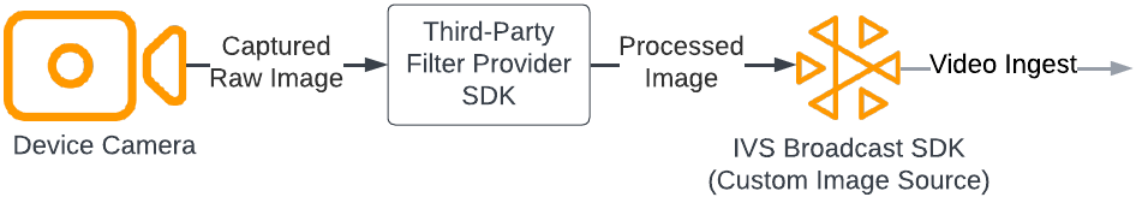
SDK de diffusion IVS : filtres de caméra tiers (Streaming en temps réel)

Ce guide part du principe que vous connaissez déjà les sources d'images personnalisées ainsi que l'intégration du SDK de diffusion en temps réel IVS dans votre application.

Les filtres de caméra permettent aux créateurs de contenu en direct d'améliorer ou de modifier l'apparence de leur visage ou de leur arrière-plan. Cela peut augmenter l'engagement des spectateurs, attirer de nouveaux spectateurs et améliorer l'expérience de diffusion en direct.

Intégration de filtres de caméra tiers

Vous pouvez intégrer des SDK de filtres de caméra tiers au SDK de diffusion IVS en transmettant la sortie du SDK de filtres à une source d'entrée d'image personnalisée. Une source d'entrée d'image personnalisée permet à une application de fournir sa propre entrée d'image au SDK de diffusion. Le SDK d'un fournisseur de filtres tiers peut gérer le cycle de vie de la caméra pour traiter les images provenant de la caméra, appliquer un effet de filtre et le produire dans un format pouvant être transmis à une source d'image personnalisée.



Consultez la documentation de votre fournisseur de filtres tiers pour connaître les méthodes intégrées qui permettent de convertir une image de caméra, avec l'effet de filtre, appliquée à un format pouvant être transmis à une source d'entrée d'image personnalisée. Le processus varie selon la version du SDK de diffusion IVS utilisée :

- Web : Le fournisseur de filtres doit pouvoir restituer sa sortie dans un élément du canevas. La méthode captureStream peut ensuite être utilisée pour renvoyer un MediaStream du contenu du canevas. Le MediaStream peut ensuite être converti en instance d'un LocalStageStream et diffusé sur une scène.
- Android : Le SDK du fournisseur de filtres peut soit restituer une image sur un appareil Android `Surface` fourni par le SDK de diffusion IVS, soit convertir l'image en bitmap. Si vous utilisez

un bitmap, il peut ensuite être rendu sur le `Surface` sous-jacent fourni par la source d'image personnalisée. Pour ce faire, vous devez le déverrouiller et l'écrire sur un canevas.

- iOS : Le SDK d'un fournisseur de filtres tiers doit fournir un cadre de caméra auquel un effet de filtre est appliqué sous forme de `CMSampleBuffer`. Consultez la documentation du SDK de votre fournisseur de filtres tiers pour savoir comment obtenir une `CMSampleBuffer` comme sortie finale après le traitement d'une image de caméra.

BytePlus

Android

Installation et configuration du SDK BytePlus Effects

Consultez le Guide d'accès à BytePlus Android pour plus de détails sur l'installation, l'initialisation et la configuration du SDK BytePlus Effects.

Configurer la source d'image personnalisée

Une fois le SDK initialisé, alimentez les images de caméra traitées avec un effet de filtre appliqué à une source d'entrée d'image personnalisée. Pour ce faire, créez une instance d'un objet `DeviceDiscovery` et créez une source d'image personnalisée. Notez que lorsque vous utilisez une source d'entrée d'image personnalisée pour le contrôle personnalisé de la caméra, le SDK de diffusion n'est plus responsable de la gestion de la caméra. Au lieu de cela, l'application est chargée de gérer correctement le cycle de vie de la caméra.

Java

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

Convertir la sortie en bitmap ainsi que le flux en source d'entrée d'image personnalisée

Pour que les images de caméra avec un effet de filtre appliqué depuis le SDK BytePlus Effect soient transférées directement vers le SDK de diffusion IVS, vous devez convertir la sortie d'une texture

du SDK BytePlus Effects en bitmap. Lorsqu'une image est traitée, la méthode `onDrawFrame()` est invoquée par le SDK. La méthode `onDrawFrame()` est une méthode publique de l'interface GLSurfaceView.renderer d'Android. Dans l'exemple d'application Android fourni par BytePlus, cette méthode est appelée sur chaque image de caméra ; la méthode produit une texture. Parallèlement, vous pouvez compléter la méthode `onDrawFrame()` par une logique qui permet de convertir cette texture en bitmap et de l'envoyer à une source d'entrée d'image personnalisée. Comme indiqué dans l'exemple de code suivant, utilisez la méthode `transferTextureToBitmap` fournie par le SDK BytePlus pour réaliser cette conversion. Cette méthode est fournie par la bibliothèque com.bytedance.labcv.core.util.ImageUtil du SDK BytePlus Effects, comme indiqué dans l'exemple de code suivant. Par la suite, vous pouvez effectuer le rendu sur l'Android sous-jacent `Surface` d'un `CustomImageSource` en écrivant le bitmap obtenu sur le canevas d'une Surface. De nombreuses invocations successives de `onDrawFrame()` donnent lieu à une séquence de bitmaps et, une fois combinées, créent un flux vidéo.

Java

```
import com.bytedance.labcv.core.util.ImageUtil;
...
protected ImageUtil imageUtility;
...


@Override
public void onDrawFrame(GL10 gl10) {
  ...
  // Convert BytePlus output to a Bitmap
  Bitmap outputBt = imageUtility.transferTextureToBitmap(output.getTexture(),ByteEffect

  Constants.TextureFormat.Texture2D,output.getWidth(), output.getHeight());

  canvas = surface.lockCanvas(null);
  canvas.drawBitmap(outputBt, 0f, 0f, null);
  surface.unlockCanvasAndPost(canvas);
```

DeepAR

Android

Consultez le Guide d'intégration Android de DeepAR pour obtenir plus de détails sur la façon d'intégrer le SDK DeepAR au SDK de diffusion Android IVS.

iOS

Consultez le Guide d’intégration iOS de DeepAR pour obtenir plus de détails sur la façon d’intégrer le SDK DeepAR au SDK de diffusion iOS IVS.

Snap

Web

Cette section suppose que vous connaissez déjà la diffusion et l’abonnement à des vidéos à l’aide du SDK de diffusion Web.

Pour intégrer le SDK Camera Kit de Snap au SDK de diffusion Web en temps réel IVS, vous devez effectuer les actions suivantes :

1. Installez le SDK Camera Kit et le Webpack. (Notre exemple utilise Webpack comme bundler, mais vous pouvez utiliser n’importe quel bundler de votre choix.)
2. Créer `index.html`.
3. Ajoutez des éléments de configuration.
4. Affichez et configurez les participants.
5. Affichez les caméras et les microphones connectés.
6. Créez une session Camera Kit.
7. Récupérez et appliquez un objectif.
8. Affichez le résultat d’une session Camera Kit sur un canevas.
9. Fournissez à Camera Kit une source multimédia pour le rendu et la diffusion d’un `LocalStageStream`.
10. Créez un fichier de configuration Webpack.

Chacune de ces étapes est décrite ci-dessous.

Installez le SDK et le Webpack du Camera Kit

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

Créez le fichier index.html

Puis, créez le modèle de code HTML et importez le SDK de diffusion Web sous forme de balise de script. Dans le code suivant, veillez à remplacer `<SDK version>` par la version du SDK de diffusion que vous utilisez.

JavaScript

```
<!--
/*! Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved. SPDX-License-
Identifier: Apache-2.0 */
-->
<!DOCTYPE html>
<html lang="en">

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

  <title>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</title>

  <!-- Fonts and Styling -->
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Roboto:300,300italic,700,700italic" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/
normalize.css" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/milligram/1.4.1/
milligram.css" />
  <link rel="stylesheet" href="./index.css" />

  <!-- Stages in Broadcast SDK -->
  <script src="https://web-broadcast.live-video.net/<SDK version>/amazon-ivs-web-
broadcast.js"></script>
</head>

<body>
  <!-- Introduction -->
  <header>
    <h1>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</h1>

    <p>This sample is used to demonstrate basic HTML / JS usage. <b><a href="https://
docs.aws.amazon.com/ivs/latest/userguide/multiple-hosts.html">Use the AWS CLI</
```

```
a></b> to create a <b>Stage</b> and a corresponding <b>ParticipantToken</b>.
 Multiple participants can load this page and put in their own tokens. You can <b><a
 href="https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#glossary"
 target="_blank">read more about stages in our public docs.</a></b></p>
  </header>
  <hr />

  <!-- Setup Controls -->

  <!-- Local Participant -->

  <hr style="margin-top: 5rem"/>

  <!-- Remote Participants -->

  <!-- Load all Desired Scripts -->

</body>

</html>
```

Ajout d'éléments de configuration

Créez le code HTML qui permet de sélectionner une caméra et un microphone ainsi que de spécifier un jeton de participant :

JavaScript

```
<!-- Setup Controls -->
  <div class="row">
    <div class="column">
      <label for="video-devices">Select Camera</label>
      <select disabled id="video-devices">
        <option selected disabled>Choose Option</option>
      </select>
    </div>
    <div class="column">
      <label for="audio-devices">Select Microphone</label>
      <select disabled id="audio-devices">
        <option selected disabled>Choose Option</option>
      </select>
    </div>
    <div class="column">
```

```
      <label for="token">Participant Token</label>
      <input type="text" id="token" name="token" />
    </div>
    <div class="column" style="display: flex; margin-top: 1.5rem">
      <button class="button" style="margin: auto; width: 100%" id="join-button">Join
 Stage</button>
    </div>
    <div class="column" style="display: flex; margin-top: 1.5rem">
      <button class="button" style="margin: auto; width: 100%" id="leave-button">Leave
 Stage</button>
    </div>
  </div>
```

Ajoutez du code HTML supplémentaire en dessous pour afficher les flux de caméra des participants locaux et distants :

JavaScript

```
 <!-- Local Participant -->
<div class="row local-container">
    <canvas id="canvas"></canvas>

    <div class="column" id="local-media"></div>
    <div class="static-controls hidden" id="local-controls">
      <button class="button" id="mic-control">Mute Mic</button>
      <button class="button" id="camera-control">Mute Camera</button>
    </div>
  </div>


  <hr style="margin-top: 5rem"/>

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

Chargez une logique supplémentaire, notamment des méthodes d'assistance pour configurer la caméra et le fichier JavaScript fourni. (Plus loin dans cette section, vous allez créer ces fichiers JavaScript et les regrouper en un seul fichier ; l'objectif est de pouvoir importer Camera Kit en tant que module. Le fichier JavaScript fourni contiendra la logique de configuration du Camera Kit, d'application d'un objectif et de diffusion du flux de caméra avec un objectif appliqué à une scène.)

JavaScript

```
<!-- Load all Desired Scripts -->
  <script src="./helpers.js"></script>
  <script src="./media-devices.js"></script>
  <!-- <script type="module" src="./stages-simple.js"></script> -->
  <script src="./dist/bundle.js"></script>
```

Affichage et configuration des participants

Ensuite, créez `helpers.js`. Il contient des méthodes d'assistance que vous utiliserez pour afficher et configurer les participants :

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;
}
```

Affichage des caméras et des microphones connectés

Ensuite, créez `media-devices.js`. Il contient des méthodes d'assistance pour afficher les caméras et les microphones connectés à votre appareil :

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,
    },
  });
}
```

Création d'une session Camera Kit

Créez `stages.js`. Il contient la logique permettant d'appliquer un objectif au flux de caméra et de diffuser le flux sur une scène. Pour la première partie de ce fichier, nous importons le SDK de diffusion et le SDK Web Camera Kit et initialisons les variables que nous utiliserons avec chaque SDK. Nous créons une session Camera Kit en appelant `createSession` après avoir démarré le SDK Web Camera Kit. Notez qu'un objet d'élément de canevas est transmis à une session ; cela indique à Camera Kit qu'il doit s'afficher dans ce canevas.

Java

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

// All helpers are expose on 'media-devices.js' and 'dom.js'
// const { setupParticipant } = window;
// const { initializeDeviceSelect, getCamera, getMic } = window;
// require('./helpers.js');
// require('./media-devices.js');

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

import {
  bootstrapCameraKit,
  createMediaStreamSource,
  Transform2D,
} from '@snap/camera-kit';

let cameraButton = document.getElementById('camera-control');
let micButton = document.getElementById('mic-control');
```

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

let controls = document.getElementById('local-controls');
let videoDevicesList = document.getElementById('video-devices');
let audioDevicesList = document.getElementById('audio-devices');

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

const liveRenderTarget = document.getElementById('canvas');

const init = async () => {
  await initializeDeviceSelect();

  const cameraKit = await bootstrapCameraKit({
    apiToken: INSERT_API_TOKEN_HERE,
  });

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

Récupération et application d’une lentille

Pour récupérer vos objectifs, insérez votre identifiant de groupe d’objectifs, qui se trouve sur le Portail des développeurs de Camera Kit. Dans cet exemple, nous appliquons simplement le premier objectif du tableau d’objectifs renvoyé.

JavaScript

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

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

Affichage de la sortie d'une session Camera Kit sur un canevas

Vous pouvez utiliser la méthode CaptureStream pour renvoyer une partie du contenu `MediaStream` du canevas. Avec un objectif appliqué, le canevas contiendra un flux vidéo provenant de la caméra. Ajoutez également des écouteurs d'événements pour les boutons qui permettent de désactiver la caméra et le microphone, ainsi que des écouteurs d'événements pour rejoindre et quitter une scène. Dans l'écouteur d'événements pour rejoindre une scène, nous faisons passer une session Camera Kit avec le `MediaStream` depuis le canevas, elle peut ainsi être diffusée sur une scène.

JavaScript

```
const snapStream = liveRenderTarget.captureStream();

  cameraButton.addEventListener('click', () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
    cameraButton.innerText = isMuted ? 'Show Camera' : 'Hide Camera';
  });

  micButton.addEventListener('click', () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? 'Unmute Mic' : 'Mute Mic';
  });

  joinButton.addEventListener('click', () => {
    joinStage(session, snapStream);
  });

  leaveButton.addEventListener('click', () => {
    leaveStage();
  });
};
```

Fournissez à Camera Kit une source multimédia pour le rendu et la diffusion d'un LocalStageStream

Pour diffuser un flux vidéo auquel un objectif est appliqué, créez une fonction appelée `setCameraKitSource` pour transmettre le `MediaStream` capturé précédemment sur le canevas. La `MediaStream` du canvas ne fonctionne pas pour le moment, car nous n'avons pas encore intégré notre flux de caméra local. Nous pouvons intégrer notre flux de caméra local en appelant la méthode d'assistance `getCamera` et en l'affectant à `localCamera`. Nous pouvons ensuite transmettre notre flux de caméra local (via `localCamera`) ainsi que l'objet de session à `setCameraKitSource`. La

fonction `setCameraKitSource` convertit notre flux de caméra local en une source multimédia pour CameraKit en appelant `createMediaStreamSource`. La source multimédia pour `CameraKit` est ensuite transformée pour refléter la caméra frontale. L'effet d'objectif est ensuite appliqué à la source multimédia et rendu sur le canevas de sortie en appelant `session.play()`.

Avec l'objectif appliqué au `MediaStream` capturé depuis le canevas, nous pouvons ensuite procéder à sa diffusion sur une scène. Pour ce faire, nous créons un `LocalStageStream` avec les pistes vidéo du `MediaStream`. Une instance de `LocalStageStream` peut ensuite être transmise à un `StageStrategy` pour être diffusée.

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;
    },
  };
```

Le code restant ci-dessous concerne la création et la gestion de notre scène :

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();
```

Création d’un fichier de configuration Webpack

Créez `webpack.config.js` et ajoutez le code suivant. Cela regroupe la logique ci-dessus pour vous puissiez utiliser l’instruction d’importation dans le but d’utiliser Camera Kit.

JavaScript

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

Enfin, exécutez `npm run build` pour regrouper votre JavaScript tel que défini dans le fichier de configuration du Webpack. Vous pouvez ensuite diffuser du code HTML et JavaScript à partir d'un serveur Web. Par exemple, vous pouvez utiliser le serveur HTTP de Python et ouvrir localhost:8000 pour voir le résultat :

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

Android

Pour intégrer le SDK Camera Kit de Snap au SDK de diffusion Android IVS, vous devez installer le SDK Camera Kit, initialiser une session Camera Kit, appliquer un objectif puis transmettre la sortie de la session Camera Kit à la source d'entrée d'image personnalisée.

Pour installer le SDK Camera Kit, ajoutez ce qui suit au fichier `build.gradle` de votre module. Remplacez `$cameraKitVersion` par la dernière version du SDK Camera Kit.

Java

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

Initialisez et obtenez un `cameraKitSession`. Camera Kit fournit également un encapsuleur pratique pour les API CameraX d'Android, ainsi, vous n'avez pas à écrire de logique compliquée pour utiliser CameraX avec Camera Kit. Vous pouvez utiliser l'objet `CameraXImageProcessorSource` comme source pour ImageProcessor, cela vous permet de démarrer le flux d'images en mode caméra.

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();
    }
```

Récupération et application des lentilles

Vous pouvez configurer les objectifs et les organiser dans le carrousel du Portail des développeurs Camera Kit :

Java

```
// Fetch lenses from repository and apply them
 // Replace LENS_GROUP_ID with Lens Group ID from https://camera-kit.snapchat.com
cameraKitSession.getLenses().getRepository().get(new Available(LENS_GROUP_ID),
 available -> {
     Log.d(TAG, "Available lenses: " + available);
     Lenses.whenHasFirst(available, lens ->
 cameraKitSession.getLenses().getProcessor().apply(lens, result -> {
          Log.d(TAG,  "Apply lens [" + lens + "] success: " + result);
      }));
});
```

Pour diffuser, envoyez des images traitées vers le `Surface` sous-jacent d'une source d'image personnalisée. Utilisez un objet `DeviceDiscovery` et créez une `CustomImageSource` pour renvoyer une `SurfaceSource`. Vous pouvez ensuite afficher le résultat d'une session `CameraKit` sur la `Surface` sous-jacente, fournie par une `SurfaceSource`.

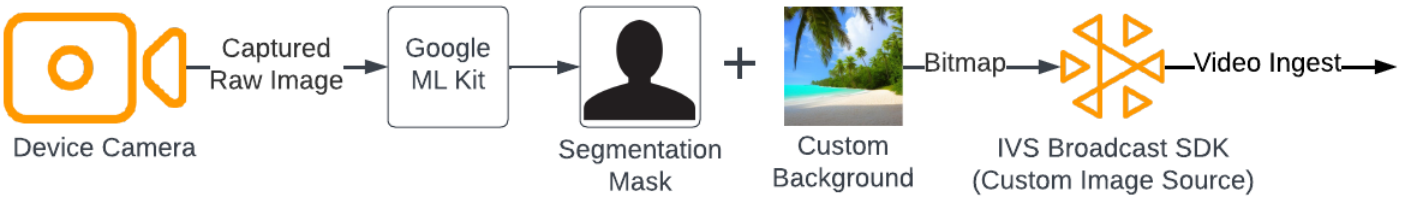
Java

```
val publishStreams = ArrayList<LocalStageStream>()

val deviceDiscovery = DeviceDiscovery(applicationContext)
val customSource =
 deviceDiscovery.createImageInputSource(BroadcastConfiguration.Vec2(720f, 1280f))

cameraKitSession.processor.connectOutput(outputFrom(customSource.inputSurface))
val customStream = ImageLocalStageStream(customSource)

// After rendering the output from a Camera Kit session to the Surface, you can
// then return it as a LocalStageStream to be published by the Broadcast SDK
val customStream: ImageLocalStageStream = ImageLocalStageStream(surfaceSource)
publishStreams.add(customStream)

@Override
fun stageStreamsToPublishForParticipant(stage: Stage, participantInfo:
 ParticipantInfo): List<LocalStageStream> = publishStreams
```

Remplacement d’arrière-plan

Le remplacement de l’arrière-plan est un type de filtre de caméra permettant aux créateurs de diffusions en direct de modifier leur arrière-plan. Comme le montre le diagramme suivant, le remplacement de votre arrière-plan implique les conditions suivantes :

1. Obtenir une image de caméra à partir du flux de caméra en direct.
2. La segmenter en composants de premier plan et d’arrière-plan à l’aide de Google ML Kit.
3. Combiner le masque de segmentation obtenu avec une image d’arrière-plan personnalisée.
4. Le transmettre à une source d’image personnalisée pour diffusion.



Web

Cette section suppose que vous connaissez déjà la diffusion et l'abonnement à des vidéos à l'aide du SDK de diffusion Web.

Pour remplacer l'arrière-plan d'une diffusion en direct par une image personnalisée, utilisez le modèle de segmentation selfie avec le segmenteur d'image MediaPipe. Il s'agit d'un modèle de machine learning qui identifie les pixels de l'image vidéo situés au premier plan ou à l'arrière-plan. Vous pouvez ensuite utiliser les résultats du modèle pour remplacer l'arrière-plan d'une diffusion en direct. Pour ce faire, copiez les pixels de premier plan du flux vidéo vers une image personnalisée représentant le nouvel arrière-plan.

Pour intégrer le remplacement en arrière-plan au SDK de diffusion Web IVS en temps réel, vous devez effectuer les actions suivantes :

1. Installez MediaPipe et Webpack. (Notre exemple utilise Webpack comme bundler, mais vous pouvez utiliser n'importe quel bundler de votre choix.)
2. Créer `index.html`.
3. Ajoutez des éléments multimédias.
4. Ajoutez une balise de script.
5. Créer `app.js`.
6. Chargez une image d'arrière-plan personnalisée.
7. Créez une instance de `ImageSegmenter`.
8. Affichez le flux vidéo sur un canevas.
9. Créez une logique de remplacement de l'arrière-plan.
10. Créez un fichier de configuration Webpack.
11. Regroupez votre fichier JavaScript.

Installation de MediaPipe et de Webpack

Pour commencer, installez les packages npm `@mediapipe/tasks-vision` et `webpack`. L'exemple ci-dessous utilise Webpack comme un bundler JavaScript, mais vous pouvez utiliser un autre bundler si vous préférez.

JavaScript

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

Assurez-vous également de mettre à jour votre script `package.json` pour spécifier `webpack` comme script de compilation :

JavaScript

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

Créez le fichier index.html

Puis, créez le modèle de code HTML et importez le SDK de diffusion Web sous forme de balise de script. Dans le code suivant, veillez à remplacer `<SDK version>` par la version du SDK de diffusion que vous utilisez.

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>
```

Ajouter des éléments multimédias

Ajoutez ensuite un élément vidéo ainsi que deux éléments de canevas dans la balise body. L'élément vidéo contiendra le flux de votre caméra en direct et servira d'entrée dans le segmenteur d'image MediaPipe. Le premier élément du canevas sert à afficher un aperçu du flux qui sera diffusé. Le deuxième élément du canevas sert à afficher l'image personnalisée qui sera utilisée comme arrière-plan. Comme le second canevas qui contient l'image personnalisée sert uniquement de source pour copier des pixels par programmation vers le canevas final, il est masqué.

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>
```

Ajout d'une balise de script

Ajoutez une balise de script pour charger un fichier JavaScript intégré qui contiendra le code permettant de remplacer l'arrière-plan et de le diffuser sur une scène :

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

Crée le fichier app.js

Créez ensuite un fichier JavaScript pour obtenir les objets des éléments du canevas ainsi que de la vidéo créés dans la page HTML. Importez les modules `ImageSegmenter` et `FilesetResolver`. Le module `ImageSegmenter` sert à effectuer la tâche de segmentation.

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";
```

Créez ensuite une fonction appelée `init()` pour récupérer le MediaStream depuis la caméra de l'utilisateur, invoquez ensuite une fonction de rappel chaque fois qu'une image de caméra finit de se charger. Ajoutez des écouteurs d'événements pour les boutons qui permettent de rejoindre et de quitter une scène.

Notez que lorsque vous rejoignez une scène, nous transmettons une variable nommée `segmentationStream`. Il s'agit d'un flux vidéo capturé à partir d'un élément du canevas. Il contient une image de premier plan superposée à l'image personnalisée représentant l'arrière-plan. Plus tard, ce flux personnalisé sera utilisé pour créer une instance d'un `LocalStageStream`, qui pourra être diffusé sur une scène.

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();
  });
};
```

Chargement d'une image d'arrière-plan personnalisée

Au bas de la fonction `init`, ajoutez du code pour appeler une fonction nommée `initBackgroundCanvas`. Cette dernière charge une image personnalisée à partir d'un fichier local et effectue un rendu sur un canevas. Nous définirons cette fonction à l'étape suivante. Assignez le `MediaStream` extrait par la caméra de l'utilisateur à l'objet vidéo. Plus tard, cet objet vidéo sera transmis au segmenteur d'image. Définissez également une fonction nommée `renderVideoToCanvas` comme fonction de rappel à invoquer chaque fois que le chargement d'une image vidéo est terminé. Nous définirons cette fonction lors d'une étape ultérieure.

JavaScript

```
initBackgroundCanvas();

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

Implémentons la fonction `initBackgroundCanvas` qui charge une image à partir d'un fichier local. Dans cet exemple, nous utilisons l'image d'une plage comme arrière-plan personnalisé. Le canevas qui contient l'image personnalisée sera masqué, car vous le fusionnerez avec les pixels de premier plan de l'élément du canevas qui contient le flux de caméra.

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);
  };
};
```

Créez une instance d'ImageSegmenter

Créez ensuite une instance de `ImageSegmenter`, qui segmentera l'image et renverra le résultat sous forme de masque. Lors de la création d'une instance d'un `ImageSegmenter`, vous utiliserez le modèle de segmentation selfie.

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,
  });
};
```

Affichage du flux vidéo sur un canevas

Créez ensuite la fonction servant à afficher le flux vidéo vers l'autre élément du canevas. Afin de pouvoir en extraire les pixels de premier plan à l'aide de l'API Canvas 2D, nous devons afficher le flux vidéo sur un canevas. Ce faisant, nous allons également transmettre une image vidéo à notre instance de `ImageSegmenter`. Pour ce faire, nous utilisons la méthode SegmentForVideo pour segmenter le premier plan par rapport à l'arrière-plan de l'image vidéo. Lorsque la méthode SegmentForVideo revient, elle invoque notre fonction de rappel personnalisée `replaceBackground` afin de remplacer l'arrière-plan.

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);
};
```

Création d’une logique de remplacement de l’arrière-plan

Créez la fonction `replaceBackground`. Cette dernière fusionne l’image d’arrière-plan personnalisée avec le premier plan du flux de caméra pour remplacer l’arrière-plan. Pour commencer, la fonction extrait les données de pixels sous-jacentes de l’image d’arrière-plan personnalisée et du flux vidéo à partir des deux éléments du canevas créés précédemment. Elle parcourt ensuite le masque fourni par `ImageSegmenter`, qui indique quels pixels se trouvent au premier plan. Au fur et à mesure que la fonction parcourt le masque, elle copie de manière sélective les pixels qui contiennent le flux de caméra de l’utilisateur vers les données de pixels d’arrière-plan correspondantes. Après cela, elle convertit les données finales en pixels avec le premier plan copié sur l’arrière-plan et les dessine sur un canevas.

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);
}
```

À titre de référence, voici le fichier `app.js` complet contenant toute la logique ci-dessus :

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();
```

Création d’un fichier de configuration Webpack

Ajoutez cette configuration à votre fichier de configuration Webpack pour regrouper `app.js`, afin que les appels d’importation fonctionnent :

JavaScript

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

Regroupement de vos fichiers JavaScript

```
npm run build
```

Démarrez un serveur HTTP simple à partir du répertoire contenant `index.html` puis ouvrez `localhost:8000` pour voir le résultat :

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

Android

Pour remplacer l'arrière-plan de votre diffusion en direct, utilisez l'API de segmentation des selfies de Google ML Kit. L'API de segmentation des selfies accepte une image de caméra en entrée et renvoie un masque qui fournit un score de confiance pour chaque pixel de l'image. Cela permet de faire la différence entre le premier plan et l'arrière-plan. Sur la base du score de confiance, vous pouvez ensuite récupérer la couleur de pixel correspondante à partir de l'image d'arrière-plan ou de celle de premier plan. Ce processus se poursuit jusqu'à ce que tous les scores de confiance du masque aient été examinés. Le résultat donne un nouveau tableau de couleurs de pixels qui contient des pixels de premier plan combinés à des pixels de l'image d'arrière-plan.

Pour intégrer le remplacement en arrière-plan au SDK de diffusion Android IVS en temps réel, vous devez effectuer les actions suivantes :

1. Installez les bibliothèques CameraX et le Google ML Kit.
2. Initialisez les variables standard.
3. Créez une source d'image personnalisée.
4. Gérez les cadres des caméras.
5. Transférez les images de la caméra au Google ML Kit.
6. Superposez le premier plan du cadre de la caméra sur votre arrière-plan personnalisé.
7. Transférez la nouvelle image à une source d'image personnalisée.

Installation des bibliothèques CameraX et de Google ML Kit

Pour extraire des images du flux de caméra en direct, utilisez la bibliothèque CameraX d'Android. Pour installer la bibliothèque CameraX et le Google ML Kit, ajoutez ce qui suit au fichier de votre module `build.gradle`. Remplacez `${camerax_version}` et `${google_ml_kit_version}` par la dernière version des bibliothèques CameraX et Google ML Kit, respectivement.

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}"
```

Importez les bibliothèques suivantes :

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
```

Initialisation des variables standard

Initialisez une instance de `ImageAnalysis` ainsi qu'une instance de `ExecutorService` :

Java

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

Initialisez une instance du segmenteur dans STREAM_MODE :

Java

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

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

Création d'une source d'image personnalisée

Dans le cadre de votre activité on`Create`, créez une instance d'un objet `DeviceDiscovery` puis créez une source d'image personnalisée. La `Surface` fournie par la source d'image personnalisée recevra l'image finale, le premier plan étant superposé à une image d'arrière-plan personnalisée.

Vous allez ensuite créer une instance d'un `ImageLocalStageStream` à l'aide de la source d'image personnalisée. L'instance d'un `ImageLocalStageStream` (nommée `filterStream` dans cet exemple) pourra ensuite être diffusée sur une scène. Consultez le Guide du SDK de diffusion Android IVS pour obtenir des instructions sur la configuration d'une scène. Enfin, créez également un fil qui sera utilisé pour gérer la caméra.

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()
```

Gestion des cadres des caméras

Créez ensuite une fonction pour initialiser la caméra. Cette fonction utilise la bibliothèque CameraX pour extraire des images du flux de caméra en direct. Tout d'abord, vous devez créer une instance d'un `ProcessCameraProvider` appelée `cameraProviderFuture`. Cet objet représente le résultat futur de l'obtention d'un fournisseur de caméras. Ensuite, chargez une image de votre projet sous forme de bitmap. Cet exemple utilise l'image d'une plage comme arrière-plan, mais il peut s'agir de l'image de votre choix.

Vous ajoutez ensuite un écouteur à `cameraProviderFuture`. Cet écouteur est averti lorsque la caméra est disponible ou si une erreur survient lors du processus d'obtention d'un fournisseur de caméra.

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))
    }
```

Dans l’écouteur, créez `ImageAnalysis.Builder` pour accéder à chaque image individuelle depuis le flux de caméra en direct. Définissez la stratégie de pression de retour sur `STRATEGY_KEEP_ONLY_LATEST`. Cela garantit qu’une seule image de caméra n’est traitée à la fois.

Convertissez chaque image de caméra en image bitmap. Cela permet d'extraire ses pixels pour les combiner ultérieurement avec l'image d'arrière-plan personnalisée.

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())
```

Transfert des images de la caméra au Google ML Kit

Ensuite, créez une `InputImage` et transmettez-la à l'instance du segmenteur pour traitement. Une `InputImage` peut être créée à partir d'un `ImageProxy` fourni par l'instance de `ImageAnalysis`. Une fois qu'une `InputImage` est fournie au segmenteur, elle renvoie un masque avec des scores de confiance qui indiquent la probabilité qu'un pixel soit au premier plan ou en arrière-plan. Ce masque fournit également des données concernant la largeur et la hauteur. Ces données serviront à créer un nouveau tableau contenant les pixels d'arrière-plan de l'image d'arrière-plan personnalisée chargée précédemment.

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)
```

Superposition du premier plan du cadre de la caméra sur votre arrière-plan personnalisé

Avec le masque contenant les scores de confiance, le cadre de la caméra sous forme de bitmap et les pixels de couleur de l'image d'arrière-plan personnalisée, vous disposez du nécessaire pour superposer le premier plan à votre arrière-plan personnalisé. La fonction `overlayForeground` est ensuite appelée avec les paramètres suivants :

Java

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

Cette fonction parcourt le masque et vérifie les valeurs de confiance. Elle permet de déterminer s'il faut obtenir la couleur de pixel correspondante à partir de l'image d'arrière-plan ou du cadre de la caméra. Si la valeur de confiance indique qu'un pixel du masque se trouve très probablement en arrière-plan, la fonction obtiendra la couleur de pixel correspondante à partir de l'image d'arrière-plan. Dans le cas contraire, elle obtiendra la couleur de pixel correspondante du cadre de la caméra pour créer le premier plan. Une fois que la fonction a fini d'itérer dans le masque, un nouveau bitmap sera créé à l'aide du nouveau tableau de pixels de couleur puis sera renvoyé. Ce nouveau bitmap contient le premier plan superposé à l'arrière-plan personnalisé.

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
        )
    }
```

Transférez la nouvelle image à une source d'image personnalisée

Vous pouvez ensuite écrire le nouveau bitmap dans une `Surface` fournie par une source d'image personnalisée. Cela diffusera le bitmap sur votre scène.

Java

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

Voici la fonction complète qui permet d'obtenir les images de la caméra, les transmettre au segmenteur et les superposer à l'arrière-plan :

Java

```
@androidx.annotation.OptIn(androidx.camera.core.ExperimentalGetImage::class)
    private fun startCamera(surface: Surface) {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
        val imageResource = R.drawable.clouds
        val bgBitmap: Bitmap = BitmapFactory.decodeResource(resources, imageResource)
        var resultBitmap: Bitmap;

        cameraProviderFuture.addListener({
            // Used to bind the lifecycle of cameras to the lifecycle owner
            val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

            val imageAnalyzer = ImageAnalysis.Builder()
            analysisUseCase = imageAnalyzer
```

```
                .setTargetResolution(Size(720, 1280))
                .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                .build()

            analysisUseCase!!.setAnalyzer(cameraExecutor) { imageProxy: ImageProxy ->
                val mediaImage = imageProxy.image
                val tempBitmap = imageProxy.toBitmap();
                val inputBitmap =
tempBitmap.rotate(imageProxy.imageInfo.rotationDegrees.toFloat())

                if (mediaImage != null) {
                    val inputImage =
                        InputImage.fromMediaImage(mediaImage,
imageProxy.imageInfo.rotationDegrees)

                    segmenter.process(inputImage)
                        .addOnSuccessListener { segmentationMask ->
                            val mask = segmentationMask.buffer
                            val maskWidth = segmentationMask.width
                            val maskHeight = segmentationMask.height
                            val backgroundPixels = IntArray(maskWidth * maskHeight)
                            bgBitmap.getPixels(backgroundPixels, 0, maskWidth, 0, 0,
maskWidth, maskHeight)

                            resultBitmap = overlayForeground(mask, maskWidth,
maskHeight, inputBitmap, backgroundPixels)
                            canvas = surface.lockCanvas(null);
                            canvas.drawBitmap(resultBitmap, 0f, 0f, null)

                            surface.unlockCanvasAndPost(canvas);

                        }
                        .addOnFailureListener { exception ->
                            Log.d("App", exception.message!!)
                        }
                        .addOnCompleteListener {
                            imageProxy.close()
                        }

                }
            };

            val cameraSelector = CameraSelector.DEFAULT_FRONT_CAMERA
```

```
            try {
                // Unbind use cases before rebinding
                cameraProvider.unbindAll()

                // Bind use cases to camera
                cameraProvider.bindToLifecycle(this, cameraSelector, analysisUseCase)

            } catch(exc: Exception) {
                Log.e(TAG, "Use case binding failed", exc)
            }

        }, ContextCompat.getMainExecutor(this))
    }
```

SDK de diffusion IVS : Modes audio mobiles (Streaming en temps réel)

La qualité audio est un élément important de toute expérience multimédia en équipe réelle. Il n'existe pas de configuration audio universelle adaptée à tous les cas d'utilisation. Pour garantir à vos utilisateurs la meilleure expérience possible lorsqu'ils écoutent un flux IVS en temps réel, nos SDK mobiles offrent plusieurs configurations audio prédéfinies, ainsi que des personnalisations plus puissantes selon les besoins.

Introduction

Les SDK de diffusion mobile IVS fournissent une classe `StageAudioManager`. Cette classe est conçue pour être le point de contact unique qui permet de contrôler les modes audio sous-jacents sur les deux plateformes. Sur Android, elle contrôle l'AudioManager, notamment le mode audio, la source audio, le type de contenu, l'utilisation et les appareils de communication. Sur iOS, elle contrôle l'application AVAudioSession et détermine si voiceProcessing est activé.

Important : N'interagissez pas avec `AVAudioSession` ou `AudioManager` directement à l'aide du SDK de diffusion en temps réel IVS ou pendant qu'il est actif. Cela pourrait entraîner des pertes de son, ou ce dernier pourrait être enregistré ou lu sur le mauvais appareil.

Avant de créer votre premier `DeviceDiscovery` ou votre premier objet `Stage`, vous devez configurer la classe `StageAudioManager`.

Android (Kotlin)

```
StageAudioManager.getInstance(context).setPreset(StageAudioManager.UseCasePreset.VIDEO_CHAT)
 The default value

val deviceDiscovery = DeviceDiscovery(context)
val stage = Stage(context, token, this)

// Other Stage implementation code
```

iOS (Swift)

```
IVSStageAudioManager.sharedInstance().setPreset(.videoChat) // The default value

let deviceDiscovery = IVSDeviceDiscovery()
let stage = try? IVSStage(token: token, strategy: self)

// Other Stage implementation code
```

Si rien n'est défini sur `StageAudioManager` avant l'initialisation d'une instance `DeviceDiscovery` ou `Stage`, le préréglage `VideoChat` s'applique automatiquement.

Préréglages de mode audio

Le SDK de diffusion en temps réel fournit trois préréglages, chacun est adapté aux cas d'utilisation courants, comme décrit ci-dessous. Pour chaque préréglage, nous traitons cinq catégories clés qui différencient les préréglages les uns des autres.

Chat vidéo

Il s'agit du préréglage par défaut. Il permet à l'appareil local d'avoir une conversation en temps réel avec les autres participants.

Catégorie	Android	iOS
Annulation de l'écho	Activées	Activées
Variateur de volume	Volume d'appel	Volume d'appel

Catégorie	Android	iOS
Sélection du microphone	Limité en fonction du système d'exploitation. Les microphon es USB peuvent ne pas être disponibles.	Limité en fonction du système d'exploitation. Les microphon es USB et Bluetooth peuvent ne pas être disponibles. Les casques Bluetooth qui gèrent à la fois les entrées et les sorties devraient fonctionner, comme les AirPods.
Sortie audio	Tout périphérique de sortie devrait fonctionner.	Limité en fonction du système d'exploitation. Les casques filaires peuvent ne pas être disponibles.
Qualité audio	Moyenne/basse. Cela ressemblera à un appel téléphonique, et non à une lecture multimédia.	Moyenne/basse. Cela ressemblera à un appel téléphonique, et non à une lecture multimédia.

S'abonner uniquement

Ce préréglage est conçu pour que vous puissiez prévoir de vous abonner à d'autres participants à la diffusion, mais que vous ne diffusez pas vous-même. Il privilégie la qualité audio et prend en charge tous les périphériques de sortie disponibles.

Catégorie	Android	iOS
Annulation de l'écho	Désactivées	Désactivées
Variateur de volume	Volume multimédia	Volume multimédia
Sélection du microphone	N/A, ce préréglage n'est pas conçu pour la diffusion.	N/A, ce préréglage n'est pas conçu pour la diffusion.
Sortie audio	Tout périphérique de sortie devrait fonctionner.	Tout périphérique de sortie devrait fonctionner.

Catégorie	Android	iOS
Qualité audio	Élevée. Tout type de média doit apparaître clairement, y compris la musique.	Élevée. Tout type de média doit apparaître clairement, y compris la musique.

Studio

Ce préréglage est conçu pour fournir un abonnement de haute qualité tout en conservant la possibilité de diffuser. Le matériel d'enregistrement et de lecture doit assurer l'annulation de l'écho. Ici, un cas d'utilisation serait de se servir d'un microphone USB et d'un casque filaire. Le SDK conservera un son de la plus haute qualité tout en s'appuyant sur la séparation physique de ces appareils pour éviter qu'ils ne génèrent de l'écho.

Catégorie	Android	iOS
Annulation de l'écho	Désactivées	Désactivées
Variateur de volume	Volume multimédia dans la plupart des cas. Volume des appels lorsqu'un microphone Bluetooth est connecté.	Volume multimédia
Sélection du microphone	N'importe quel microphone devrait fonctionner.	N'importe quel microphone devrait fonctionner.
Sortie audio	Tout périphérique de sortie devrait fonctionner.	Tout périphérique de sortie devrait fonctionner.
Qualité audio	Élevée. Les deux parties doivent être en mesure d'envoyer de la musique et de l'entendre clairemen t de l'autre extrémité. Lorsqu'un casque Bluetooth est connecté, la qualité audio baisse en raison de l'activation du mode Bluetooth SCO.	Élevée. Les deux parties doivent être en mesure d'envoyer de la musique et de l'entendre clairemen t de l'autre extrémité. Lorsqu'un casque Bluetooth est connecté, la qualité audio peut baisser en raison de l'activation du

Catégorie	Android	iOS
		mode Bluetooth SCO, en fonction du casque.

Cas d'utilisation avancés

Outre les préréglages, les SDK de diffusion en temps réel iOS et Android permettent de configurer les modes audio de la plateforme sous-jacente :

- Sur Android, définissez les paramètres AudioSource, Usage et ContentType.
- Sur iOS, utilisez AVAudioSession.Category, AVAudioSession.CategoryOptions, AVAudioSession.mode et la possibilité de choisir si le traitement vocal est activé ou non lors de la diffusion.

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
```

Diffusion via Bluetooth sur Android

Sur Android, le SDK revient automatiquement au préréglage `VIDEO_CHAT` lorsque les conditions suivantes sont remplies :

- La configuration attribuée n'utilise pas la valeur d'utilisation `VOICE_COMMUNICATION`.
- Un microphone Bluetooth est connecté à l'appareil.
- Le participant local diffuse sur une scène.

Il s'agit d'une limitation du système d'exploitation Android en ce qui concerne la manière dont les casques Bluetooth sont utilisés pour l'enregistrement audio.

Intégration avec d'autres SDK

Comme iOS et Android ne prennent en charge qu'un seul mode audio actif par application, des conflits peuvent survenir si votre application utilise plusieurs SDK qui nécessitent le contrôle du mode audio. Lorsque vous êtes confronté à de tels conflits, il existe des stratégies de résolution courantes à essayer ; elles sont expliquées ci-dessous.

Faites correspondre les valeurs du mode audio

À l'aide des options de configuration audio avancées du SDK IVS ou des fonctionnalités de l'autre SDK, faites en sorte que les deux SDK s'alignent sur les valeurs sous-jacentes.

Agora

iOS

Sur iOS, le fait de demander au SDK Agora de garder la `AVAudioSession` active l'empêchera de se désactiver pendant que le SDK de diffusion en temps réel IVS l'utilise.

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

Android

Évitez d’appeler `setEnableSpeakerphone` sur `RtcEngine` ; appelez `enableLocalAudio(false)` pendant la diffusion grâce au SDK de diffusion en temps réel IVS. Vous pouvez appeler `enableLocalAudio(true)` à nouveau lorsque le SDK IVS ne diffuse pas.

Utilisation d’Amazon EventBridge avec le streaming en temps réel IVS

Vous pouvez utiliser Amazon EventBridge pour contrôler vos flux Amazon Interactive Video Service (IVS).

Amazon IVS envoie des événements de modification sur l’état de vos flux à Amazon EventBridge. Tous les événements fournis sont valides. Cependant, les événements sont envoyés sur la base du meilleur effort, ce qui signifie qu’il n’y a aucune garantie que :

- Les événements sont livrés : un événement désigné peut se produire (par exemple, un participant a publié), mais il est possible qu’Amazon IVS n’envoie pas d’événement correspondant à EventBridge. Amazon IVS essaie de livrer des événements pendant plusieurs heures avant d’abandonner.
- Les événements qui sont livrés arriveront dans un délai spécifié : vous pouvez recevoir des événements datant de quelques heures.
- Les événements sont livrés dans l’ordre : les événements peuvent être envoyés dans le désordre, surtout s’ils sont envoyés à peu de temps d’intervalle. Par exemple, vous pouvez voir un participant dépublié avant qu’il ne soit diffusé.

Bien qu’il soit rare que des événements soient manquants, en retard ou en désordre, vous devez prendre en compte ces éventualités si vous écrivez des programmes stratégiques qui dépendent de l’ordre ou de l’existence des événements de notification.

Vous pouvez créer des règles EventBridge pour tous les événements suivants.

Type d’événement	Événement	Envoyé lorsque …
Changements d’état de montage IVS	Échec de la destination	Une tentative de sortie vers une destination a échoué. Par exemple, la diffusion sur une chaîne a échoué, car il n’y avait pas de clé de flux ou parce qu’une autre diffusion était en cours.

Type d’événement	Événement	Envoyé lorsque …
Changements d’état de montage IVS	Début de la destination	La sortie vers une destination a démarré avec succès.
Changements d’état de montage IVS	Fin de la destination	La sortie vers une destination est terminée.
Changements d’état de montage IVS	Reconnexion à une destination	La sortie vers une destination a été interrompue et une tentative de reconnexion est en cours.
Changements d’état de montage IVS	Début de la session	Une session de montage a été créée. Cet événement se déclenche lorsqu’un pipeline de processus de montage s’initialise avec succès. À ce stade, le pipeline de montage s’est inscrit avec succès à une scène, reçoit du contenu multimédi a et est en mesure de composer une vidéo.
Changements d’état de montage IVS	Fin de la session	Une session de montage terminée.
Changements d’état de montage IVS	Échec de la session	Un pipeline de montage n’a pas pu être initialisé en raison de l’indispo nibilité des ressources de scène ou d’une autre erreur interne.
Mise à jour de scène IVS	Participant diffusé	Un participant commence à être diffusé sur une étape.
Mise à jour de scène IVS	Participant non diffusé	Un participant a cessé d’être diffusé sur une scène.

Création de règles Amazon EventBridge pour Amazon IVS

Vous pouvez créer une règle qui se déclenche sur un événement émis par Amazon IVS. Suivez les étapes décrites dans Create a rule in Amazon EventBridge dans le Guide de l'utilisateur d'Amazon EventBridge. Lorsque vous devez sélectionner un service, choisissez Interactive Video Service (IVS).

Exemples : changements d'état de montage

Échec de la destination : cet événement est envoyé lorsqu'une tentative de sortie vers une destination a échoué. Par exemple, la diffusion sur une chaîne a échoué, car il n'y avait pas de clé de flux ou parce qu'une autre diffusion était en cours.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Destination Failure",
     "stage_arn": "<stage-arn>",
     "id": "<Destination-id>",
     "reason": "eg. stream key invalid"
   }
}
```

Début de la destination : cet événement est envoyé lorsque la sortie vers une destination démarre avec succès.

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

Fin de la destination : cet événement est envoyé lorsque la sortie vers une destination est terminée.

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

Reconnexion de la destination : cet événement est envoyé lorsque la sortie vers une destination a été interrompue et qu'une tentative de reconnexion est en cours.

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

Début de session : cet événement est envoyé lors de la création d'une session de montage. Cet événement se déclenche lorsqu'un pipeline de processus de montage s'initialise avec succès. À ce stade, le pipeline de montage s'est inscrit avec succès à une scène, reçoit du contenu multimédia et est en mesure de composer une vidéo.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session Start",
     "stage_arn": "<stage-arn>"
   }
}
```

Fin de session : cet événement est envoyé lorsqu'une session de montage est terminée et que toutes les ressources ont été supprimées.

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

Échec de session : cet événement est envoyé lorsqu'un pipeline de montage n'a pas pu être initialisé en raison de l'indisponibilité des ressources de la scène, de l'absence de participants ou de toute autre erreur interne.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session Failure",
     "stage_arn": "<stage-arn>",
     "reason": "eg. no participants in the stage"
   }
}
```

Exemples : mise à jour d'étape

Les événements de mise à jour d'étape incluent un nom d'événement (qui classe l'événement) et des métadonnées relatives à l'événement. Les métadonnées incluent l'ID du participant qui a déclenché l'événement, les ID d'étape et de session associés, ainsi que l'ID utilisateur.

Participant diffusé : cet événement est envoyé lorsqu'un participant commence à être diffusé sur une étape.

```
{
```

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

Participant non diffusé : cet événement est envoyé lorsqu'un participant a cessé d'être diffusé sur une étape.

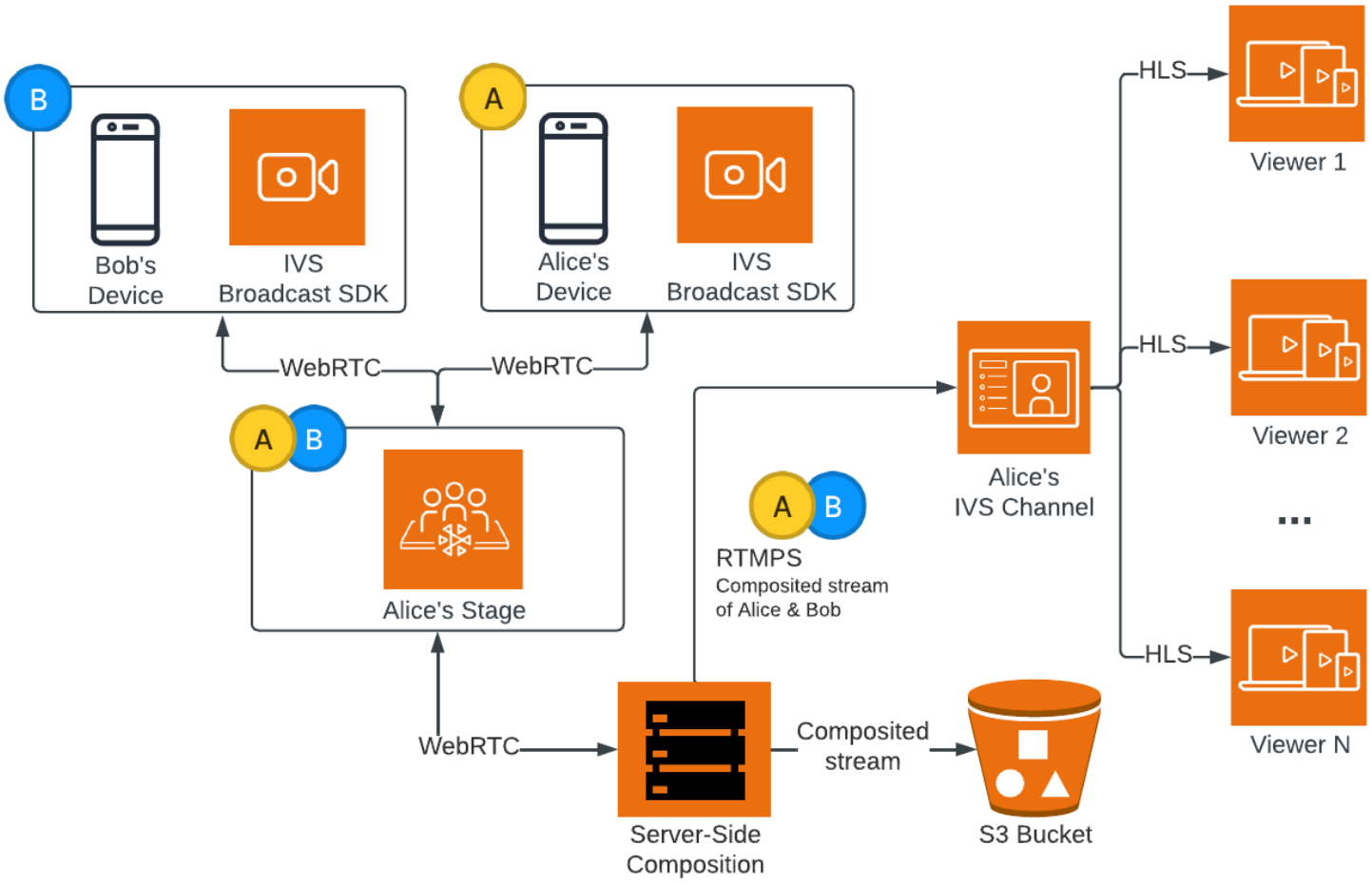
```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"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

Montage côté serveur (Streaming en temps réel)

Le montage côté serveur utilise un serveur IVS pour mixer le son et la vidéo de tous les participants à la scène, puis envoie cette vidéo mixée à un canal IVS (par exemple, pour atteindre un public plus large) ou à un compartiment S3. Le montage côté serveur est invoqué via les points de terminaison du plan de contrôle IVS situés dans la région d'origine de la scène.

Diffuser ou enregistrer une scène à l'aide d'un montage côté serveur offre de nombreux avantages, ce qui en fait un choix intéressant pour les utilisateurs qui recherchent des flux de travail vidéos efficaces et fiables sur le cloud.

Ce schéma illustre le fonctionnement du montage côté serveur :



Avantages

Par rapport au montage côté client, le montage côté serveur présente les avantages suivants :

- Réduction de la charge client : Grâce au montage côté serveur, la charge du traitement et de la combinaison des sources audio et vidéo est transférée des appareils clients individuels au serveur lui-même. Le montage côté serveur évite aux appareils clients d'utiliser leur processeur et leurs ressources réseau pour composer la vue et la transmettre à IVS. Cela signifie que les spectateurs peuvent regarder l'émission sans que leurs appareils aient à gérer des tâches gourmandes en ressources. Cela peut améliorer l'autonomie de la batterie ainsi que les expériences de visionnage.
- Qualité constante : La montage côté serveur offre un contrôle précis de la qualité, de la résolution et du débit du flux final. Cela garantit une expérience visuelle cohérente pour tous les spectateurs, quelles que soient les capacités de leurs appareils individuels.
- Résilience : En centralisant le processus de montage sur le serveur, la diffusion devient plus robuste. Même si l'appareil d'un diffuseur de publication présente des limites ou des fluctuations techniques, le serveur peut s'adapter et fournir un flux plus fluide à l'ensemble du public.
- Efficacité de la bande passante : Puisque le serveur gère le montage, les diffuseurs de publication de scènes n'ont pas à dépenser de bande passante supplémentaire pour diffuser la vidéo sur IVS.

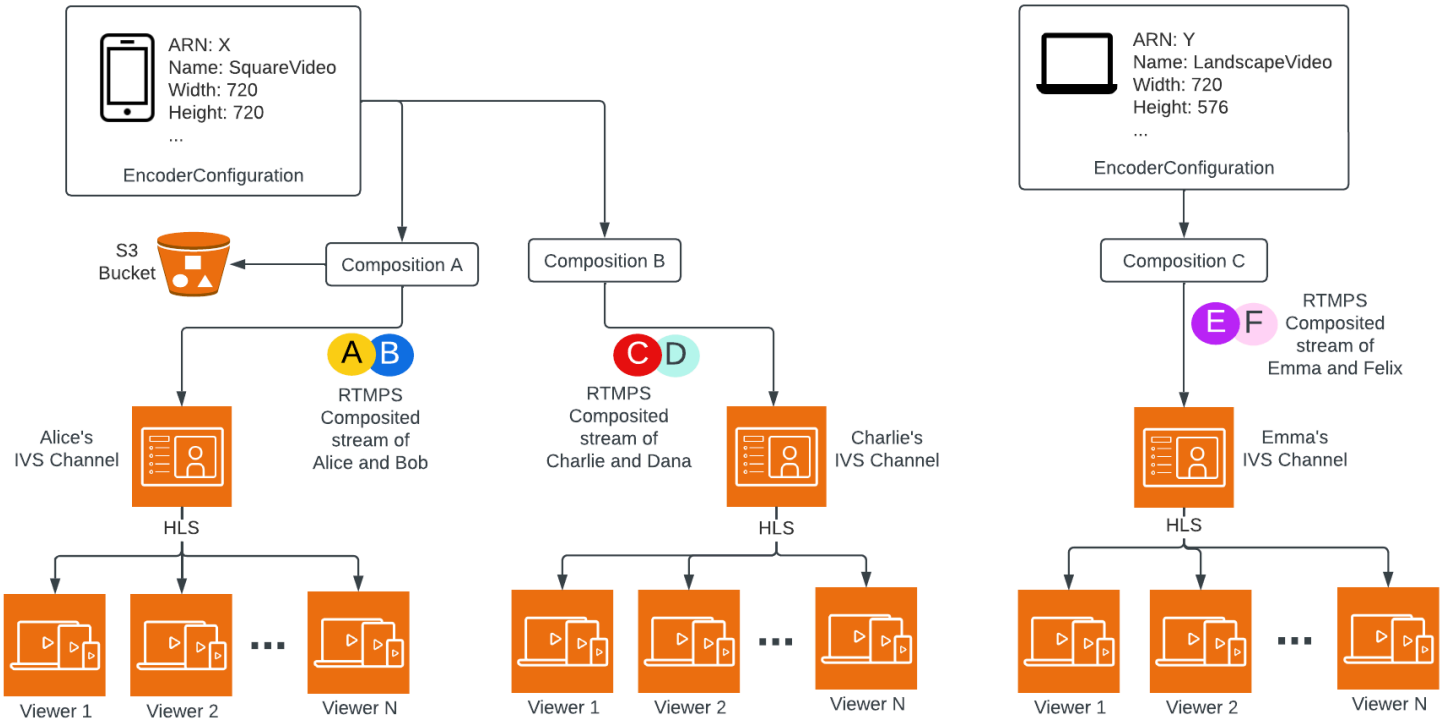
Pour diffuser une scène sur une chaîne IVS, vous pouvez également effectuer le montage côté client ; consultez Activation de plusieurs hôtes sur un flux IVS dans le Guide de l'utilisateur du streaming IVS à faible latence.

API IVS

Le montage côté serveur utilise les éléments clés de l'API suivants :

- Un objet EncoderConfiguration vous permet de personnaliser le format de la vidéo à générer (hauteur, largeur, débit et autres paramètres de diffusion). Vous pouvez réutiliser un objet EncoderConfiguration chaque fois que vous appelez le point de terminaison StartComposition.
- Les points de terminaison de montage suivent le montage vidéo et le transmettent sur un canal IVS.
- StorageConfiguration suit le compartiment S3 dans lequel les montages sont enregistrés.

Pour utiliser le montage côté serveur, vous devez créer un objet EncoderConfiguration et l'attacher lorsque vous appelez le point de terminaison StartComposition. Dans cet exemple, l'objet EncoderConfiguration SquareVideo est utilisé dans deux montages :

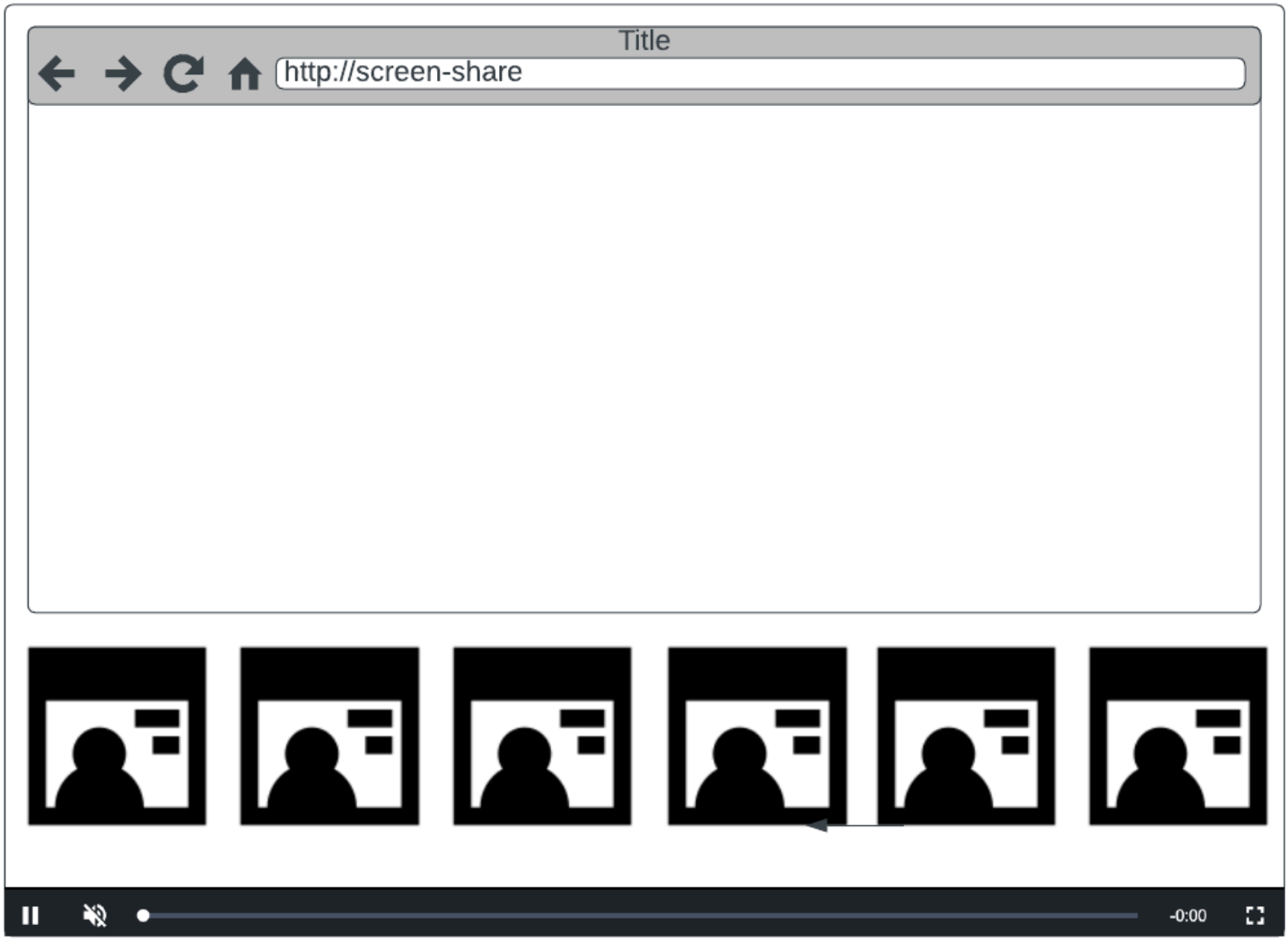


Pour obtenir des informations complètes, consultez la [Référence de l'API de streaming en temps réel Amazon IVS](#).

Layouts

Par défaut, la fonction de montage côté serveur utilise une disposition en grille pour organiser les participants à la scène dans des créneaux de taille égale :

Cette disposition permet aux clients de configurer et d'invoquer un emplacement en vedette. L'emplacement sélectionné se trouve sur l'écran principal, tandis que les autres participants apparaissent en dessous dans des emplacements de même taille :



Remarque : La résolution maximale prise en charge par un diffuseur de publication de scènes pour les montages côté serveur est de 1080p. Si un diffuseur de publication envoie une vidéo d'une résolution supérieure à 1080p, il sera affiché en tant que participant audio uniquement.

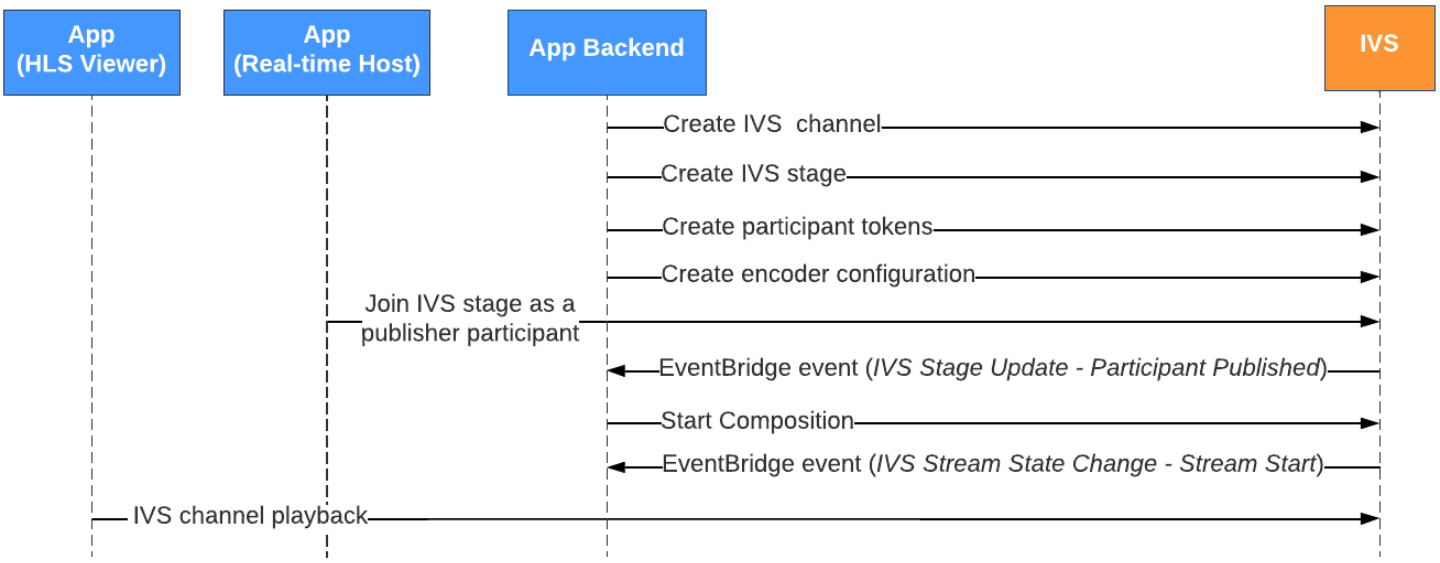
Démarrer

Prérequis

Pour utiliser le montage côté serveur, vous devez disposer d'une scène avec des diffuseurs de publication actifs et utiliser un canal IVS et/ou un compartiment S3 comme destination de montage. Nous décrivons ci-dessous un flux de travail possible qui utilise les événements EventBridge pour démarrer un montage qui diffuse la scène sur une chaîne IVS lorsqu'un participant diffuse. Vous pouvez également démarrer et arrêter des montages en fonction de la logique de votre propre

application. Voir Enregistrement composite pour un autre exemple illustrant l'utilisation d'un montage côté serveur pour enregistrer une scène directement dans un compartiment S3.

1. Créez un canal IVS. Consultez Démarrer avec la diffusion à faible latence d'Amazon IVS.
2. Créez une scène IVS et des jetons de participation pour chaque diffuseur de publication.
3. Créez un objet EncoderConfiguration.
4. Joignez-vous à la scène et diffusez dessus. (Consultez les sections « Diffusion et abonnement » des guides du SDK de diffusion en temps réel : Web, Android et iOS.)
5. Lorsque vous recevez un événement EventBridge diffusé par un participant, appelez StartComposition.
6. Patientez quelques secondes pour voir la vue composite apparaître lors de la lecture des chaînes.



Remarque : Un montage s'arrête automatiquement après 60 secondes d'inactivité de la part du diffuseur de publication participant à la scène. À ce stade, le montage prend fin et passe à un état `STOPPED`. Un montage est automatiquement supprimé après quelques minutes passées dans l'état `STOPPED`.

Instructions de la CLI

L'utilisation de l'AWS CLI est une option avancée qui nécessite le téléchargement et la configuration de la CLI sur votre machine. Pour plus de détails, consultez le Guide de l'utilisateur de l'Interface de ligne de commande AWS.

Vous pouvez désormais utiliser l'interface de ligne de commande pour créer et gérer des ressources. Les points de terminaison du montage se trouvent sous l'espace de noms `ivs-realtime`.

Création de la ressource EncoderConfiguration

Une ressource EncoderConfiguration vous permet de personnaliser le format de la vidéo à générer (hauteur, largeur, débit et autres paramètres de diffusion). Vous pouvez réutiliser une ressource EncoderConfiguration chaque fois que vous appelez le point de terminaison Montage, comme expliqué à l'étape suivante.

La commande ci-dessous crée une ressource EncoderConfiguration qui configure les paramètres de montage vidéo côté serveur, tels que le débit vidéo, la fréquence d'images et la résolution :

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

La réponse est :

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

Commencer un montage

À l'aide de l'ARN EncoderConfiguration fourni dans la réponse ci-dessus, créez votre ressource de montage :

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/9W59OBY2M8s4"}}]'
```

La réponse indiquera que le montage est créé avec un état `STARTING`. Une fois que le Montage commence à diffuser la composition, l'état passe à `ACTIVE`. (Vous pouvez consulter l'état en appelant le point de terminaison ListCompositions ou GetComposition.)

Une fois qu'un montage est `ACTIVE`, la vue composite de la scène IVS est visible sur le canal IVS, à l'aide de ListCompositions :

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

La réponse est :

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

Remarque : Pour que le montage reste actif, les diffuseurs de publication participants doivent diffuser activement sur la scène. Pour plus d'informations, consultez les sections « Diffusion et abonnement » des guides du SDK de diffusion en temps réel : Web, Android et iOS. Vous devez créer un jeton de scène distinct pour chaque participant.

Activer le partage d'écran

Pour utiliser une disposition de partage d'écran fixe, suivez les étapes indiquées ci-dessous.

Création de la ressource EncoderConfiguration

La commande ci-dessous crée une ressource EncoderConfiguration qui configure les paramètres de montage côté serveur (débit vidéo, fréquence d'images et résolution).

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

Créez un jeton de participant à la scène avec un attribut `screen-share`. Comme nous allons spécifier `screen-share` en tant que nom de l'emplacement `featured`, nous devons créer un jeton de scène dont l'attribut `screen-share` est défini sur `true` :

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

La réponse est :

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

Commencer le montage

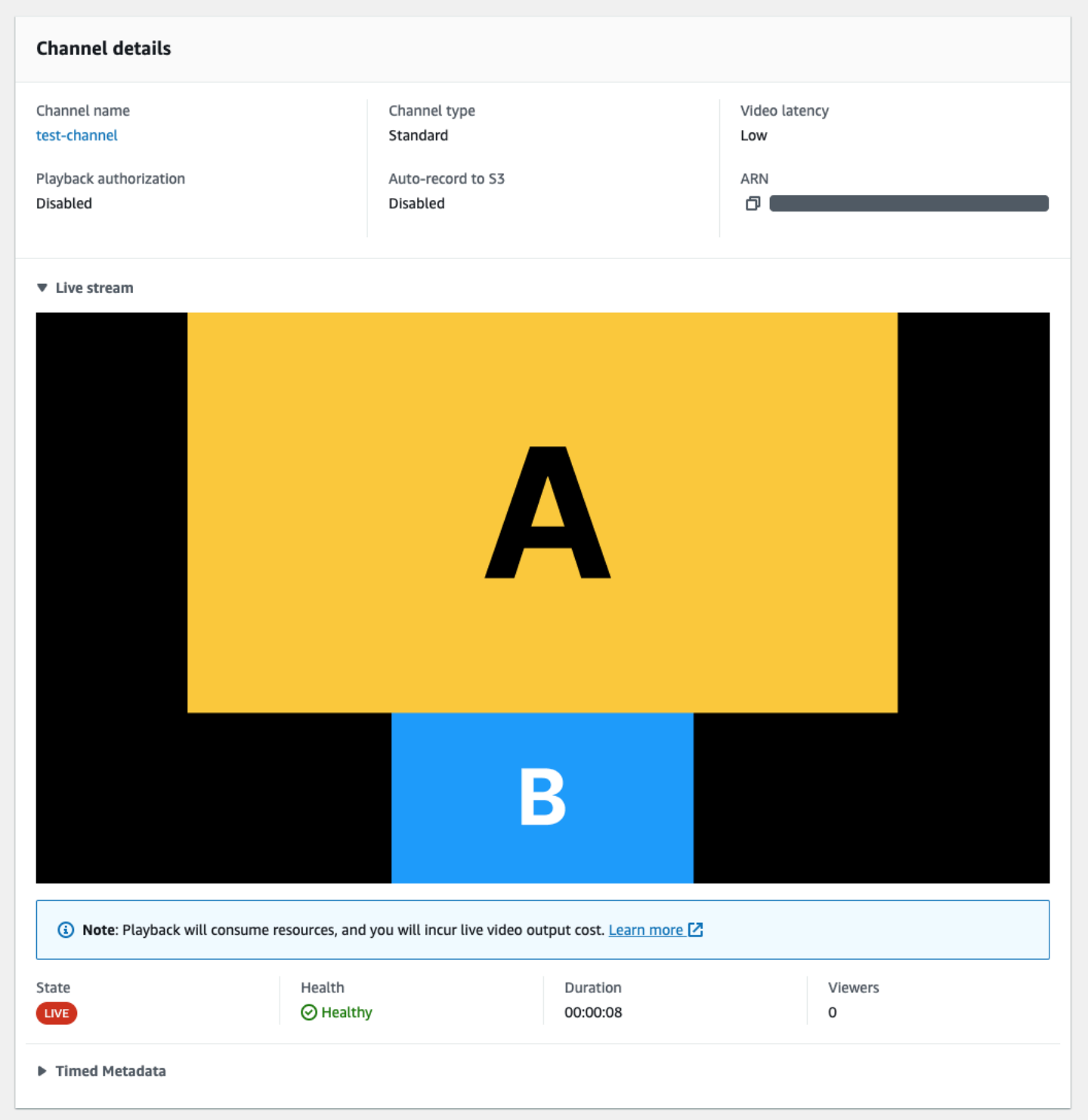
Pour démarrer le montage à l'aide de la fonctionnalité de partage d'écran, nous utilisons cette commande :

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVaOwO"}}]' --layout
 "grid={featuredParticipantAttribute=screen-share}"
```

La réponse est :

```
{
   "composition" : {
      "arn" : "arn:aws:ivs:us-east-1:927810967299:composition/B19tQcXRgtoz",
      "destinations" : [ {
         "configuration" : {
            "channel" : {
               "channelArn" : "arn:aws:ivs:us-east-1:927810967299:channel/
DOlMW4dfMR8r",
               "encoderConfigurationArn" : "arn:aws:ivs:us-east-1:927810967299:encoder-
configuration/DEkQHWPVaOwO"
            },
            "name" : ""
         },
         "id" : "SGmgBXTULuXv",
         "state" : "STARTING"
      } ],
      "layout" : {
         "grid" : {
            "featuredParticipantAttribute" : "screen-share"
         }
      },
      "stageArn" : "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
      "startTime" : "2023-09-27T21:32:38Z",
      "state" : "STARTING",
      "tags" : { }
   }
}
```

Lorsque le participant `E813MFklPWLF` rejoint la scène, la vidéo de ce participant sera affichée dans l'emplacement en vedette, et tous les autres diffuseurs de publication d'scène seront affichés en dessous de l'emplacement :



Arrêter le montage

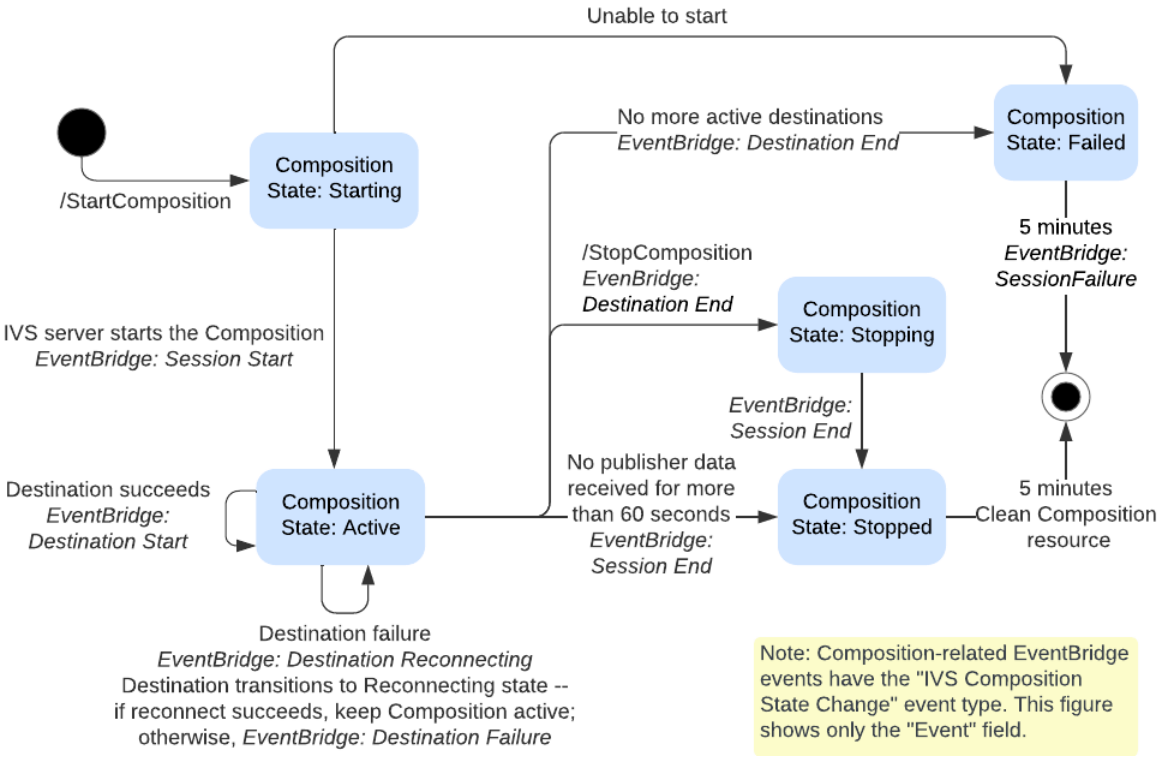
Pour arrêter un montage à tout moment, appelez le point de terminaison StopComposition :

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

Cycle de vie des montages

Utilisez le schéma ci-dessous pour comprendre les transitions d'état d'un montage. Au niveau général, le cycle de vie d'un montage est le suivant :

1. Une ressource de montage est créée lorsque l'utilisateur appelle le point de terminaison StartComposition
2. Une fois qu'IVS démarre correctement le montage, un événement EventBridge « Changement d'état du montage IVS (début de session) » est envoyé. Consultez Utilisation d'Amazon EventBridge avec le streaming en temps réel IVS pour plus de détails sur les événements.
3. Une fois qu'un montage est actif, les événements suivants peuvent se produire :
   - L'utilisateur arrête le montage : Si le point de terminaison StopComposition est appelé, IVS initie un arrêt progressif du montage en envoyant des événements de « fin de destination » suivis d'un événement de « fin de session ».
   - Le montage s'arrête automatiquement : Si aucun participant ne publie activement sur la scène IVS, le montage est finalisé automatiquement au bout de 60 secondes et les événements EventBridge sont envoyés.
   - Défaillance de destination : Si une destination échoue de manière inattendue (par exemple, le canal IVS est supprimé), la destination passe à l'état `RECONNECTING` et un événement de « reconnexion de destination » est envoyé. Si la restauration est impossible, IVS fait passer la destination à l'état `FAILED` et un événement « Échec de la destination » est envoyé. IVS maintient le montage en fonctionnement si au moins une de ses destinations est active.
4. Une fois que le montage est à l'état `STOPPED` ou `FAILED`, il est automatiquement effacé au bout de cinq minutes. (Le cas échéant, il ne peut plus être récupéré par ListCompositions ou GetComposition.)


Unable to start
/StartComposition
Composition State: Starting
No more active destinations
EventBridge: Destination End
Composition State: Failed
5 minutes
EventBridge: SessionFailure
/StopComposition
EvenBridge:
Destination End
Composition State: Stopping
IVS server starts the Composition
EventBridge: Session Start
EventBridge: Session End
Destination succeeds
EventBridge: Destination Start
Composition State: Active
No publisher data received for more than 60 seconds
EventBridge: Session End
Composition State: Stopped
5 minutes
Clean Composition resource
Destination failure
EventBridge: Destination Reconnecting
Destination transitions to Reconnecting state -- if reconnect succeeds, keep Composition active; otherwise, EventBridge: Destination Failure
Note: Composition-related EventBridge events have the "IVS Composition State Change" event type. This figure shows only the "Event" field.

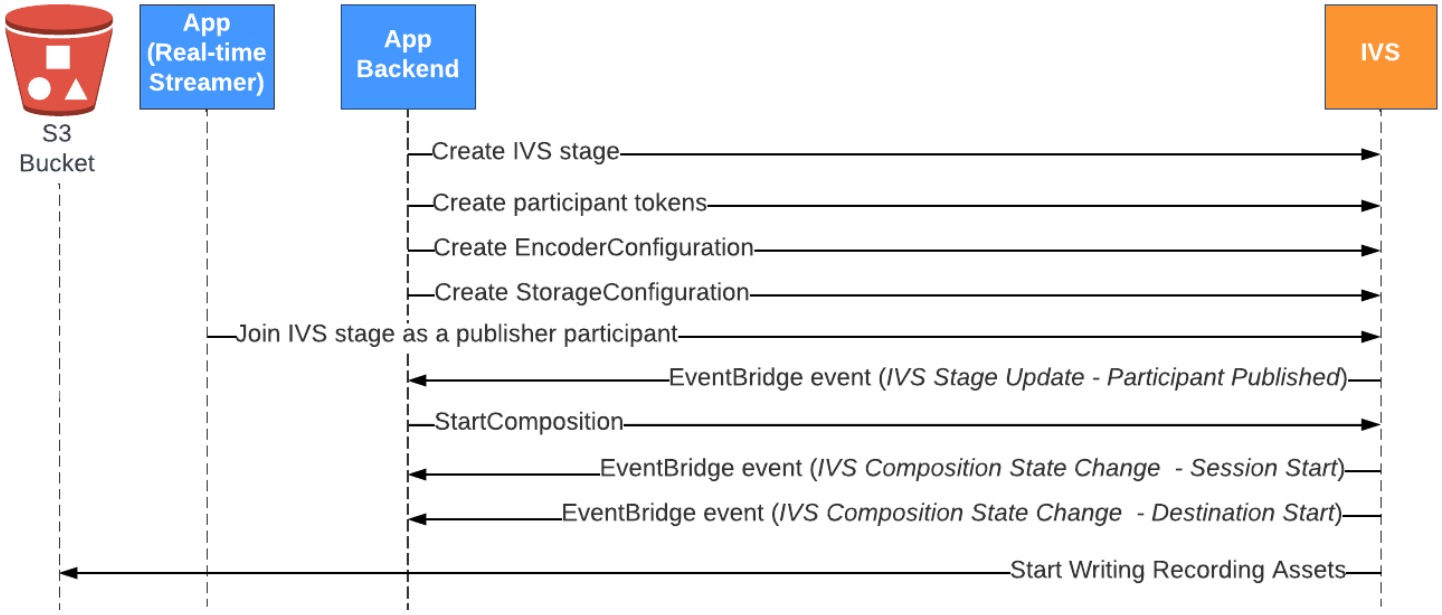
Enregistrement composite (diffusion en temps réel)

Ce document explique comment utiliser la fonction d'enregistrement composite dans un montage côté serveur. L'enregistrement composite vous permet de générer des enregistrements HLS d'une scène IVS. Pour ce faire, il combine efficacement tous les diffuseurs de publication en une seule vue à l'aide d'un serveur IVS, puis enregistre la vidéo obtenue dans un compartiment S3.

Prérequis

Pour utiliser l'enregistrement composite, vous devez disposer d'une scène avec des éditeurs actifs et d'un compartiment S3 à utiliser comme destination d'enregistrement. Nous décrivons ci-dessous un flux de travail possible qui utilise EventBridge des événements pour enregistrer une composition dans un compartiment S3. Vous pouvez également démarrer et arrêter des montages en fonction de la logique de votre propre application.

1. Créez une scène IVS et des jetons de participation pour chaque diffuseur de publication.
2. Créez un EncoderConfiguration(un objet représentant la manière dont la vidéo enregistrée doit être rendue).
3. Créez un compartiment S3 et un StorageConfiguration(où le contenu de l'enregistrement sera stocké).
4. Joignez-vous à la scène et diffusez dessus.
5. Lorsque vous recevez un EventBridge événement publié par un participant, appelez StartCompositionavec un DestinationConfiguration objet S3 comme destination
6. Au bout de quelques secondes, vous devriez être en mesure de voir que les segments HLS sont conservés dans vos compartiments S3.



Remarque : un montage s'arrête automatiquement après 60 secondes d'inactivité de la part du diffuseur de publication participant à la scène. À ce stade, le montage prend fin et passe à un état `STOPPED`. Un montage est automatiquement supprimé après quelques minutes passées dans l'état `STOPPED`. Pour plus de détails, consultez la section Cycle de vie des montages dans Montage côté serveur.

Exemple d'enregistrement composite : StartComposition avec une destination de compartiment S3

L'exemple ci-dessous montre un appel typique au StartCompositionpoint de terminaison, spécifiant S3 comme seule destination pour la composition. Une fois que le montage passe à un état `ACTIVE`, les segments vidéo et les métadonnées commencent à être écrits dans le compartiment S3 spécifié par l'objet `storageConfiguration`. Pour créer des montages avec différentes mises en page, consultez la rubrique « Mises en page » de la section Montage côté serveur ainsi que la Référence de l'API de diffusion en temps réel IVS.

Demande

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

{
   "destinations": [
      {
```

```
         "s3": {
            "encoderConfigurationArns": [
              "arn:aws:ivs:ap-northeast-1:927810967299:encoder-configuration/
PAAwglkRtjge"
            ],
            "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq"
         }
      }
   ],
   "idempotencyToken": "db1i782f1g9",
   "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr"
}
```

Réponse

```
{
    "composition": {
        "arn": "arn:aws:ivs:ap-northeast-1:927810967299:composition/s2AdaGUbvQgp",
        "destinations": [
            {
                "configuration": {
                    "name": "",
                    "s3": {
                        "encoderConfigurationArns": [
                            "arn:aws:ivs:ap-northeast-1:927810967299:encoder-
configuration/PAAwglkRtjge"
                        ],
                        "recordingConfiguration": {
                            "format": "HLS"
                        },
                        "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq"
                    }
                },
                "detail": {
                    "s3": {
                        "recordingPrefix": "MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/
composite"
                    }
                },
                "id": "2pBRKrNgX1ff",
                "state": "STARTING"
```

```
            }
        ],
        "layout": null,
        "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr",
        "startTime": "2023-11-01T06:25:37Z",
        "state": "STARTING",
        "tags": {}
    }
}
```

Le `recordingPrefix` champ présent dans la StartComposition réponse peut être utilisé pour déterminer où le contenu de l'enregistrement sera stocké.

Enregistrer des contenus

Lorsque la composition passe à un `ACTIVE` état, vous commencez à voir des segments vidéo HLS et des fichiers de métadonnées être écrits dans le compartiment S3 fourni lors de l'appel StartComposition. Ce contenu sont disponibles pour le post-traitement ou la lecture en tant que vidéo à la demande.

Notez qu'une fois qu'un montage est mis en ligne, un événement « IVS Composition State Change » est émis et l'écriture des fichiers manifestes ainsi que des segments vidéo peut être rapide. Nous vous recommandons de ne lire ou de ne traiter les flux enregistrés qu'après réception de l'événement « Changement d'état du montage IVS (fin de session) ». Pour plus de détails, voir Utilisation EventBridge avec le streaming en temps réel IVS.

Voici un exemple de structure de répertoire et de contenu d'un enregistrement d'une session IVS en direct :

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

Le dossier `events` contient les fichiers de métadonnées correspondant à l'événement d'enregistrement. Les fichiers de métadonnées JSON sont générés lorsque l'enregistrement démarre, se termine avec succès ou se termine avec des échecs :

- `events/recording-started.json`
- `events/recording-ended.json`
- `events/recording-failed.json`

Un fichier `events` contiendra `recording-started.json` et `recording-ended.json` ou `recording-failed.json`.

Ceux-ci contiennent des métadonnées relatives à la session enregistrée et à ses formats de sortie. Les détails JSON sont fournis ci-dessous.

Le dossier `media` contient le contenu multimédia pris en charge. Le sous-dossier `hls` contient tous les fichiers multimédia et manifestes générés au cours de la session de montage et est lisible avec le lecteur Amazon IVS. Le manifeste HLS se trouve dans le dossier `multivariant.m3u8`.

Politique relative aux compartiments pour StorageConfiguration

Lorsqu'un StorageConfiguration objet est créé, IVS aura accès pour écrire du contenu dans le compartiment S3 spécifié. Cet accès est accordé en apportant des modifications à la politique du compartiment S3. Si la politique du compartiment est modifiée de manière à supprimer l'accès d'IVS, les enregistrements en cours et les nouveaux enregistrements échoueront.

L'exemple ci-dessous montre une politique de compartiment S3 qui permet à IVS d'écrire dans le compartiment S3 :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CompositeWrite-y1d212y",
            "Effect": "Allow",
            "Principal": {
                "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::my-s3-bucket/*",
            "Condition": {
```

```
                "StringEquals": {
                    "s3:x-amz-acl": "bucket-owner-full-control"
                },
                "Bool": {
                    "aws:SecureTransport": "true"
                }
            }
        }
    ]
}
```

Fichiers de métadonnées JSON

Ces métadonnées sont au format JSON. Il contient les informations suivantes :

Champ	Type	Obligatoire	Description
`stage_arn`	chaîne	Oui	ARN de la scène utilisée comme source du montage.
`media`	objet	Oui	Objet contenant les objets énumérés du contenu multimédia disponibles pour cet enregistrement. Valeurs valides : `"hls"`.
`hls`	objet	Oui	Champ énuméré qui décrit la sortie du format Apple HLS.
`duration_ms`	entier	Conditionnel	Durée du contenu HLS enregistré en millisecondes. Celle-ci n'est disponible que lorsque `recording_status` est `"RECORDING_ENDED"` ou `"RECORDING_ENDED_WITH_FAILURE"`. Si un échec s'est produit avant l'enregistrement, la valeur est 0.

Champ	Type	Obligatoire	Description
`path`	chaîne	Oui	Chemin relatif depuis le préfixe S3 où le contenu HLS est stocké.
`playlist`	chaîne	Oui	Nom du fichier de la liste de lecture principale HLS.
`renditions`	objet	Oui	Tableau des rendus (variante HLS) d'objets de métadonnées. Il y a toujours au moins un rendu.
`path`	chaîne	Oui	Chemin relatif depuis le préfixe S3 où le contenu HLS est stocké pour ce rendu.
`playlist`	chaîne	Oui	Nom du fichier de la liste de lecture des médias pour ce rendu.
`resolution_height`	ent	Conditionnel	Hauteur de résolution en pixels de la vidéo encodée. Cette option n'est disponible que lorsque le rendu contient une piste vidéo.
`resolution_width`	ent	Conditionnel	Largeur de résolution en pixels de la vidéo encodée. Cette option n'est disponible que lorsque le rendu contient une piste vidéo.

Champ	Type	Obligatoire	Description
`recording_ended_at`	chaîne	Conditionnel	Horodatage RFC 3339 UTC à la fin de l'enregistrement. Disponible uniquement lorsque `recording_status` est `"RECORDING_ENDED"` ou `"RECORDING_ENDED_WITH_FAILURE"`. `recording_started_at` et `recording_ended_at` sont des horodatages lorsque ces événements sont générés et peuvent ne pas correspondre exactement aux horodatages du segment vidéo HLS. Pour déterminer avec précision la durée d'un enregistrement, utilisez le champ `duration_ms`.
`recording_started_at`	chaîne	Conditionnel	Horodatage RFC 3339 UTC lorsque l'enregistrement a démarré. Ceci n'est pas disponible quand `recording_status` est à l'état `RECORDING_START_FAILED`. Veuillez lire la note ci-dessus relative à `recording_ended_at`.

Champ	Type	Obligatoire	Description
`recording_status`	chaîne	Oui	Statut de l'enregistrement. Valeurs valides: `"RECORDING_STARTED"`, `"RECORDING_ENDED"`, `"RECORDING_START_FAILED"`, `"RECORDING_ENDED_WITH_FAILURE"`.
`recording_status_message`	chaîne	Conditionnel	Informations descriptives sur le statut. Disponible uniquement lorsque `recording_status` est `"RECORDING_ENDED"` ou `"RECORDING_ENDED_WITH_FAILURE"`.
`version`	chaîne	Oui	Version du schéma des métadonnées.

Exemple : 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
        }
      ]
```

```
    }
  }
}
```

Exemple : 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
        }
      ]
    }
  }
}
```

Exemple : 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
        }
      ]
    }
  }
}
```

Lecture de contenu enregistré à partir de compartiments privés

Par défaut, le contenu enregistré est privé ; par conséquent, ces objets sont inaccessibles à la lecture à l'aide de l'URL directe S3. Si vous essayez d'ouvrir la liste de lecture multivariée HLS (fichier m3u8) pour la lecture à l'aide du lecteur IVS ou d'un autre lecteur, vous obtiendrez une erreur (par exemple, « Vous n'avez pas l'autorisation d'accéder à la ressource demandée »). Vous pouvez plutôt lire ces fichiers avec le CloudFront CDN (Content Delivery Network) Amazon.

CloudFront les distributions peuvent être configurées pour diffuser du contenu à partir de buckets privés. Cela est généralement préférable à des compartiments librement accessibles où les lectures contournent les commandes proposées par CloudFront. Vous pouvez configurer votre distribution pour qu'elle soit servie à partir d'un compartiment privé en créant un contrôle d'accès à l'origine (OAC), qui est un CloudFront utilisateur spécial doté d'autorisations de lecture sur le compartiment d'origine privé. Vous pouvez créer l'OAC après avoir créé votre distribution, via la CloudFront console ou l'API. Consultez la section Création d'un nouveau contrôle d'accès à l'origine dans le manuel Amazon CloudFront Developer Guide.

Configuration de la lecture CloudFront avec CORS activé

Cet exemple explique comment un développeur peut configurer une CloudFront distribution avec CORS activé, permettant ainsi la lecture de ses enregistrements depuis n'importe quel domaine. Cela est particulièrement utile pendant la phase de développement. N'hésitez pas à modifier l'exemple ci-dessous pour l'adapter à vos besoins de production.

Étape 1 : Créer un compartiment S3

Créez un compartiment S3 qui sera utilisé pour stocker les enregistrements. Notez que le compartiment doit se trouver dans la même région que celle que vous utilisez pour votre flux de travail IVS.

Ajouter une poilitque d'autorisation CORS au compartiment :

1. Dans la console AWS, accédez à l'onglet Autorisations du compartiment S3.
2. Copiez la stratégie CORS ci-dessous et collez-la sous Partage de ressources inter-origines (CORS). Cela permettra d'activer l'accès CORS sur le compartiment S3.

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

Étape 2 : Création d'une CloudFront distribution

Consultez la section Création d'une CloudFront distribution dans le guide du CloudFront développeur.

Dans la console AWS, entrez les informations suivantes :

Pour ce champ …	Choisissez ceci...
Domaine de l’origine	Compartiment S3 que vous avez créé à l’étape précédente
Accès à l’origine	Paramètres de contrôle d’accès à l’origine (recommandés)
Comportement du cache par défaut : politique de protocole du visualiseur	Redirect HTTP to HTTPS
Comportement du cache par défaut : méthodes HTTP autorisées	GET, HEAD et OPTIONS
Comportement de cache par défaut : clé de cache et demandes d’origine	CachingDisabled politique
Comportement du cache par défaut : politique de demande d’origine	CORS-S3Origin
Comportement du cache par défaut : politique des en-têtes de réponse	SimpleCORS
Pare-feu d’application Web	Activez les protections de sécurité

Enregistrez ensuite la CloudFront distribution.

Étape 3 : Configurer la stratégie de compartiment S3

1. Supprimez tout StorageConfiguration ce que vous avez configuré pour le compartiment S3. Cela supprimera toutes les politiques de compartiment qui ont été automatiquement ajoutées au moment de la création de la politique pour ce compartiment.
2. Accédez à votre CloudFront distribution, assurez-vous que tous les champs de distribution sont dans les états définis à l'étape précédente, puis copiez la politique de compartiment (utilisez le bouton Copier la politique).
3. Accédez à votre compartiment S3. Dans l’onglet Autorisations, sélectionnez Modifier la politique de compartiment puis collez la politique de compartiment que vous avez copiée à l’étape

précédente. Après cette étape, la politique de compartiment doit inclure cette CloudFront politique exclusivement.

4. Créez un StorageConfiguration, en spécifiant le compartiment S3.

Une fois StorageConfiguration le créé, vous verrez deux éléments dans la politique du compartiment S3, l'un permettant CloudFront de lire le contenu et l'autre permettant à IVS d'écrire du contenu. Un exemple de politique de compartiment finale, avec CloudFront accès IVS, est présenté dans Exemple : Stratégie de compartiment S3 avec CloudFront accès IVS.

Étape 4 : Lire des enregistrements

Après avoir correctement configuré la CloudFront distribution et mis à jour la politique relative aux compartiments, vous devriez être en mesure de lire les enregistrements à l'aide du lecteur IVS :

1. Démarrez correctement un montage et assurez-vous qu'un enregistrement est stocké dans le compartiment S3.
2. Après avoir suivi les étapes 1 à 3 de cet exemple, les fichiers vidéo devraient être disponibles pour la consommation via l' CloudFront URL. Votre CloudFront URL est le nom du domaine de distribution dans l'onglet Détails de la CloudFront console Amazon. Elle doit ressembler à ceci :

   ```
   a1b23cdef4ghij.cloudfront.net
   ```

3. Pour lire la vidéo enregistrée via la CloudFront distribution, recherchez la clé d'objet de votre `multivariant.m3u8` fichier dans le compartiment s3. Elle doit ressembler à ceci :

   ```
   FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/
   multivariant.m3u8
   ```

4. Ajoutez la clé de l'objet à la fin de votre CloudFront URL. Votre URL finale ressemblera à ce qui suit :

   ```
   https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/
   fjFKbylPb3k4/composite/media/hls/multivariant.m3u8
   ```

5. Vous pouvez désormais ajouter l'URL finale à l'attribut source d'un lecteur IVS pour visionner l'enregistrement complet. Pour visionner la vidéo enregistrée, vous pouvez utiliser la démonstration fournie dans la section Démarrer du Guide Web : SDK du lecteur IVS.

Exemple : politique de compartiment S3 avec CloudFront accès IVS

L'extrait ci-dessous illustre une politique de compartiment S3 qui permet CloudFront de lire le contenu dans le compartiment privé et IVS d'écrire du contenu dans le compartiment. Remarque : ne faites pas un copier/coller de l'extrait ci-dessous dans votre propre compartiment. Votre politique doit contenir les identifiants correspondant à votre CloudFront distribution et StorageConfiguration.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CompositeWrite-7eiKaIGkC9DO",
      "Effect": "Allow",
      "Principal": {
        "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        },
        "Bool": {
          "aws:SecureTransport": "true"
        }
      }
    },
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::844311324168:distribution/
E1NG4YMW5MN25A"
        }
```

```
      }
    }
  ]
}
```

Résolution des problèmes

- La composition n'est pas écrite dans le compartiment S3 : assurez-vous que le compartiment S3 et les StorageConfiguration objets sont créés et se trouvent dans la même région. Assurez-vous également qu'IVS a accès au bucket en vérifiant votre politique de bucket ; voir Bucket Policy pour StorageConfiguration.
- Je ne trouve pas de composition lorsque je l'interprète ListCompositions. Les compositions sont des ressources éphémères. Une fois qu'ils sont passés à un état final, ils sont automatiquement supprimés au bout de quelques minutes.
- Mon montage s'arrête automatiquement : un montage s'arrête automatiquement si aucun diffuseur de publication n'est présent sur la scène pendant plus de 60 secondes.

Problème connu

La liste de lecture multimédia écrite par enregistrement composite possède la balise `#EXT-X-PLAYLIST-TYPE:EVENT` pendant que le montage est en cours. Lorsque le montage est terminé, la balise est mise à jour vers `#EXT-X-PLAYLIST-TYPE:VOD`. Pour une expérience de lecture fluide, nous vous recommandons de n'utiliser cette liste de lecture qu'une fois le montage terminé avec succès.

Support OBS et WHIP (diffusion en temps réel)

Ce document explique comment utiliser des encodeurs compatibles WHIP tels que OBS pour publier sur le streaming en temps réel IVS. Le WHIP (WebRTC-HTTP Ingestion Protocol) est un projet de l'IETF développé pour normaliser l'ingestion du WebRTC.

WHIP assure la compatibilité avec des logiciels tels que OBS, offrant une alternative (au SDK de diffusion IVS) pour la publication assistée par ordinateur. Les streamers plus sophistiqués qui connaissent bien OBS peuvent le préférer en raison de ses fonctionnalités de production avancées, telles que les transitions de scène, le mixage audio et la superposition des graphiques. Les développeurs disposent ainsi d'une option polyvalente : utiliser le SDK de diffusion Web IVS pour la publication directe par navigateur ou autoriser les streamers à utiliser OBS sur leur bureau pour obtenir des outils plus puissants.

De plus, WHIP est utile dans les situations où l'utilisation du SDK de diffusion IVS n'est pas faisable ou préférée. Par exemple, dans les configurations impliquant des encodeurs matériels, le SDK de diffusion IVS peut ne pas être une option. Toutefois, si l'encodeur prend en charge le protocole WHIP, vous pouvez toujours publier directement depuis l'encodeur vers IVS.

Guide OBS

OBS prend en charge le WHIP à partir de la version 30. Pour commencer, téléchargez OBS v30 ou version ultérieure : https://obsproject.com/.

Pour publier sur un stage IVS à l'aide d'OBS via WHIP, procédez comme suit :

1. Générez un jeton de participant doté d'une fonctionnalité de publication. En termes de WHIP, un jeton de participant est un jeton porteur. Par défaut, les jetons des participants expirent au bout de 12 heures, mais vous pouvez prolonger leur durée jusqu'à 14 jours.
2. Cliquez sur Paramètres. Dans la section Stream du panneau Paramètres, sélectionnez WHIP dans le menu déroulant Service.
3. Pour le serveur, entrez https://global.whip.live-video.net/.
4. Pour le jeton porteur, entrez le jeton participant que vous avez généré à l'étape 2.
5. Configurez vos paramètres vidéo comme vous le feriez normalement, avec quelques restrictions :
   a. Le streaming en temps réel IVS prend en charge les entrées jusqu'à 720p à 8,5 Mbps. Si vous dépassez l'une de ces limites, votre diffusion sera déconnectée.

b. Nous vous recommandons de régler l'intervalle entre les images-clés dans le panneau de sortie sur 1 s ou 2 s. Un faible intervalle entre les images-touches permet aux spectateurs de démarrer la lecture vidéo plus rapidement. Nous recommandons également de régler le préréglage d'utilisation du processeur sur ultrarapide et sur Tune sur zéro latence, afin de réduire au maximum le temps de latence.

c. OBS ne prenant pas en charge la diffusion simultanée, nous vous recommandons de maintenir votre débit inférieur à 2,5 Mbits/s. Cela permet aux spectateurs utilisant des connexions à faible bande passante de regarder.

6. Appuyez sur Démarrer le streaming.

Quotas de service (Streaming en temps réel)

Voici les quotas de service et les limites pour les points de terminaison, les ressources et autres opérations d'Amazon Interactive Video Service (IVS) temps réel. Les quotas de service, également appelés limites, représentent le nombre maximal de ressources ou d'opérations de service pour votre compte AWS. Autrement dit, ces limites s'entendent par compte AWS, sauf indication contraire dans la table. Consultez également la section AWS Quotas de service.

Pour vous connecter par programmation à un service AWS, vous utilisez un point de terminaison. Consultez également la section AWS Service Endpoints.

Tous les quotas sont appliqués par région.

Augmentations des Service Quotas

Pour les quotas ajustables, vous pouvez demander une augmentation de taux via la console AWS. Vous pouvez également utiliser la console pour afficher des informations sur les Service Quotas.

Les quotas de taux d'appels API ne sont pas réglables.

Quotas de taux d'API

Type de point de terminaison	Point de terminaison	Par défaut
Montage	`GetComposition`	5 TPS
Montage	`ListCompositions`	5 TPS
Montage	`StartComposition`	5 TPS
Montage	`StopComposition`	5 TPS
Encodeur multimédia	`CreateEncoderConfiguration`	5 TPS
Encodeur multimédia	`DeleteEncoderConfiguration`	5 TPS
Encodeur multimédia	`GetEncoderConfiguration`	5 TPS
Encodeur multimédia	`ListEncoderConfigurations`	5 TPS

Type de point de terminaison	Point de terminaison	Par défaut
Étape	`CreateParticipantToken`	50 TPS
Étape	`CreateStage`	5 TPS
Étape	`DeleteStage`	5 TPS
Étape	`DisconnectParticipant`	5 TPS
Étape	`GetParticipant`	5 TPS
Étape	`GetStage`	5 TPS
Étape	`GetStageSession`	5 TPS
Étape	`ListStages`	5 TPS
Étape	`UpdateStage`	5 TPS
Étape	`ListParticipants`	5 TPS
Étape	`ListParticipantEvents`	5 TPS
Étape	`ListStageSessions`	5 TPS
StorageConfiguration	`CreateStorageConfiguration`	5 TPS
StorageConfiguration	`DeleteStorageConfiguration`	5 TPS
StorageConfiguration	`GetStorageConfiguration`	5 TPS
StorageConfiguration	`ListStorageConfigurations`	5 TPS
Balises	`ListTagsForResource`	10 TPS
Balises	`TagResource`	10 TPS
Balises	`UntagResource`	10 TPS

Autres quotas

Ressource ou fonction	Par défaut	Ajustable	Description
Configurations de l’encodeur	20	Oui	Nombre maximum de ressources de configuration de l’encodeur par compte.
Destinations de montage	2	Non	Nombre maximal d’objets de destination dans une ressource de montage.
Montage : durée maximale	24	Non	Durée maximale pendant laquelle un montage peut exister, en heures.
Montages	5	Oui	Nombre maximal de ressources de montage simultanées par compte.
Durée de publication ou d’abonnement du participant	24	Non	Durée maximale, en heures, pendant laquelle un participant peut publier ou rester abonné à une étape.
Résolution de publication du participant	720p	Non	Résolution maximale de la vidéo publiée par les participants.
Débit de téléchargement du participant	8,5 Mbit/s	Non	Débit de téléchargement agrégé maximal pour tous les abonnements d’un participant.
Participants à l’étape (diffuseurs de publication)	12	Non	Nombre maximum de participants pouvant être diffusés simultanément à une étape.
Participants à l’étape (abonnés)	10 000	Oui	Nombre maximum de participants pouvant être abonnés simultanément à une étape.

Ressource ou fonction	Par défaut	Ajustable	Description
Étapes	100	Oui	Nombre maximal de scènes par Région AWS.

Optimisations du streaming en temps réel

Pour garantir à vos utilisateurs la meilleure expérience possible lorsqu'ils diffusent et visionnent des vidéos à l'aide du streaming en temps réel IVS, il existe plusieurs manières d'améliorer ou d'optimiser certaines parties de l'expérience, en utilisant les fonctionnalités que nous proposons aujourd'hui.

Introduction

Lors de l'optimisation de la qualité de l'expérience d'un utilisateur, il est important de prendre en compte l'expérience souhaitée, qui peut changer en fonction du contenu qu'il regarde et des conditions du réseau.

Tout au long de ce guide, nous nous concentrons sur les utilisateurs qui sont soit des diffuseurs de publication de flux, soit des abonnés à des flux, et nous prenons en compte les actions et expériences souhaitées par ces utilisateurs.

Streaming adaptatif : encodage en couches avec Simulcast

Cette fonctionnalité n'est prise en charge que dans les versions de client suivantes :

- iOS et Android 1.12.0+
- Web 1.5.1+

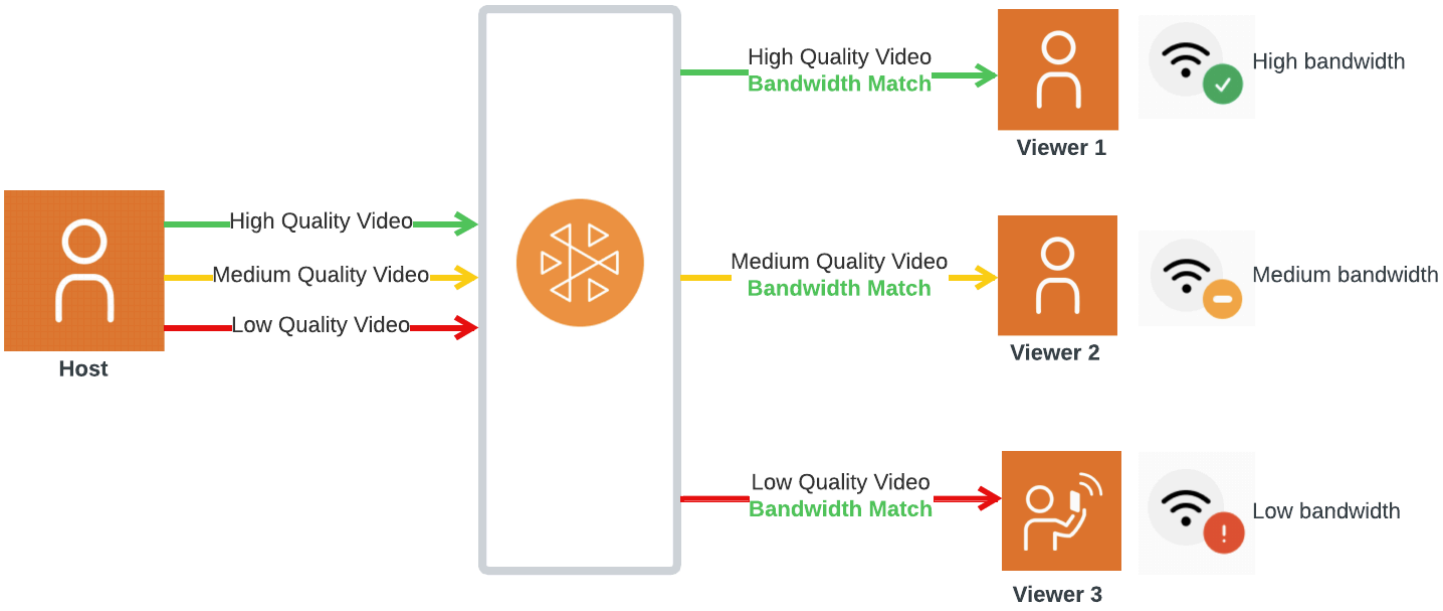
Vous devez envoyer un e-mail à amazon-ivs-simulcast@amazon .com pour activer cette fonctionnalité pour votre compte. Si vous n'êtes pas inscrit, l'activation de la diffusion simultanée via la configuration du SDK n'aura aucun effet.

Un fois que vous avez opté pour a fonctionnalité, lors de l'utilisation des SDK de diffusion IVS temps réel, les diffuseurs de publication encodent plusieurs couches de vidéos et les abonnés s'adaptent ou modifient automatiquement la qualité en fonction de leur réseau. C'est ce que nous appelons l'encodage en couches avec simulcast.

L'encodage en couches avec simulcast est pris en charge sur Android et iOS, ainsi que sur les navigateurs de bureau Chrome (pour Windows et macOS). Nous ne prenons pas en charge l'encodage en couches sur les autres navigateurs.

Dans le schéma ci-dessous, l'hôte envoie trois qualités vidéo (haute, moyenne et faible). IVS transmet la vidéo de la plus haute qualité à chaque spectateur en fonction de la bande passante

disponible fournissant ainsi une expérience optimale à chaque spectateur. Si la connexion réseau du spectateur 1 passe de bonne à mauvaise, IVS commence automatiquement à envoyer au spectateur 1 une vidéo de moindre qualité, afin que le spectateur 1 puisse continuer à regarder le flux sans interruption (avec la meilleure qualité possible).



Couches, qualités et fréquences d'images par défaut

Les qualités et les couches par défaut proposées aux utilisateurs mobiles et Web sont les suivantes :

Mobile (Android, iOS)	Web (Chrome)
Couche haute (ou personnalisée) : • Débit max. : 900 000 bits/s • Fréquence d'images : 15 images par seconde • Résolution : 360 x 640	Couche haute (ou personnalisée) : • Débit max. : 1 700 000 bps • Fréquence d'images : 30 images par seconde • Résolution : 1280 x 720
Couche intermédiaire : aucune (inutile, car la différence entre les débits de couche haute et basse sur mobiles est minime)	Couche intermédiaire : • Débit max. : 700 000 bits/s • Fréquence d'images : 20 images par seconde • Résolution : 640 x 360
Couche basse :	Couche basse :

Mobile (Android, iOS)	Web (Chrome)
• Débit max. : 150 000 bits/s • Fréquence d'images : 15 images par seconde • Résolution : 180 x 320	• Débit max. : 200 000 bits/s • Fréquence d'images : 15 images par seconde • Résolution : 320 x 180

Configuration de l'encodage en couches avec Simulcast

Pour utiliser le codage en couches avec la diffusion simultanée, vous devez avoir opté pour cette fonctionnalité et l'avoir activée sur le client. Si vous l'activez, vous constaterez une augmentation du débit global transmis, avec l'avantage de réduire le gel des vidéos.

Android

```
// Opt-out of Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
// Opt-out of Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

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

// Other Stage implementation code
```

Web

```
// Opt-out of Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})
```

```
// Other Stage implementation code
```

Configurations de streaming

Cette section explore d'autres configurations que vous pouvez apporter à vos flux vidéo et audio.

Modification du débit binaire du flux

Pour modifier le débit de votre flux vidéo, utilisez les exemples de configuration suivants.

Android

```
StageVideoConfiguration config = new StageVideoConfiguration();

// Update Max Bitrate to 1.5mbps
config.setMaxBitrate(1500000);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamVideoConfiguration();

// Update Max Bitrate to 1.5mbps
try! config.setMaxBitrate(1500000);

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

// Other Stage implementation code
```

Web

```
// Note: On web it is also recommended to configure the framerate of your device from
 userMedia
const camera = await navigator.mediaDevices.getUserMedia({
   video: {
      bitrate: {
         ideal: 1500,
         max: 1500,
```

```
      },
   },
});

let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Bitrate to 1.5mbps or 1500kbps
   maxBitrate: 1500
})

// Other Stage implementation code
```

Modification de la fréquence d'images du flux vidéo

Pour modifier la fréquence d'images de votre flux vidéo, utilisez les exemples de configuration suivants.

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
```

Optimisation du débit binaire audio et du support stéréo

Pour modifier le débit binaire et les paramètres stéréo de votre flux audio, utilisez les exemples de configuration suivants.

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
```

Optimisations suggérées

Scénario	Recommandations
Diffusion en continu avec du texte ou du contenu se déplaçant lentement , comme des présentations ou des diapositives	Vous pouvez utiliser le codage en couches avec la diffusion simultanée ou configurez des flux avec une fréquence d'images inférieure.
Diffusions avec de l'action ou beaucoup de mouvement	Vous pouvez utiliser le codage en couches avec la diffusion simultanée.
Diffusions avec des dialogues ou peu de mouvement	Vous pouvez utiliser le codage en couches avec la diffusion simultanée ou choisir l'audio uniquement (voir « S'abonner aux participants » dans les guides du SDK de diffusion en temps réel : Web, Android et iOS).
Utilisateurs diffusant en continu avec des données limitées	Vous pouvez utiliser le codage en couches avec la diffusion simultanée ou, si vous souhaitez réduire la

Scénario	Recommandations
	consommation de données pour tout le monde, configurez une fréquence d'images inférieure et réduisez le débit manuellement.

Ressources et assistance (Real-Time Streaming)

Ressources

https://ivs.rocks/ est un site dédié permettant de parcourir du contenu publié (démonstrations, exemples de code, articles de blog), d'estimer les coûts et de découvrir Amazon IVS via des démonstrations en direct.

Démonstrations

La démo d'IVS Real-Time Streaming pour iOS et Android montre aux développeurs comment utiliser Amazon IVS pour créer une application de contenu en temps réel attrayante générée par les utilisateurs sociaux. Cette application propose un flux défilant de flux en temps réel générés par les utilisateurs. Les utilisateurs peuvent créer des flux vidéo et des salles exclusivement audio. Les invités peuvent participer au streaming vidéo en mode spot invité ou en mode versus (VS).

Les instructions relatives au déploiement du backend requis et à la création de l'application sont disponibles dans les référentiels GitHub suivants :

- iOS : https://github.com/aws-samples/amazon-ivs-real-time-for-ios-demo/
- Android :https://github.com/aws-samples/amazon-ivs-real-time-for-android-demo/
- Backend : https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

Support

Le Centre de support AWS offre une large gamme de plans qui vous permettent d'accéder à des outils et au savoir-faire nécessaires pour prendre en charge vos solutions AWS. Tous les plans de support offrent un accès 24 h/24, 7 j/7 au service client. Pour accéder au support technique et à d'autres ressources qui vous aideront à planifier, déployer et améliorer votre environnement AWS, choisissez le plan de support le plus adapté à votre utilisation d'AWS.

AWS Premium Support est un canal d'assistance individuel, à temps de réponse rapide vous permettant de concevoir et d'exécuter des applications sur AWS.

AWS re:Post est un site communautaire de questions-réponses qui permet aux développeurs de discuter de questions techniques liées à Amazon IVS.

Contactez-nous comporte des liens pour les questions non techniques sur votre compte ou votre facturation. Pour les questions techniques, veuillez utiliser les forums de discussion ou les liens de support ci-dessus.

Glossaire

Consultez également le glossaire AWS. Dans le tableau ci-dessous, LL signifie streaming à faible latence IVS, tandis que RT signifie streaming en temps réel IVS.

Terme	Description	LL	RT	Chat
AAC	Codage audio avancé. L'AAC est une norme de codage audio pour la compression audio numérique avec perte. Conçu pour succéder au format MP3, le format AAC permet généralement d'obtenir une meilleure qualité sonore que le MP3 avec un débit identique. L'AAC a été normalisé par l'ISO et la CEI dans le cadre des spécifications MPEG-2 et MPEG-4.	✓	✓	
Streaming à débit binaire adaptatif	Le streaming à débit binaire adaptatif (ABR) permet au lecteur IVS de passer à un débit binaire inférieur lorsque la qualité de la connexion diminue, et de revenir à un débit supérieur lorsqu'elle s'améliore.	✓		
Streaming adaptatif	Consultez Encodage en couches avec Simulcast.		✓	
Utilisateur administratif	Utilisateur AWS disposant d'un accès administratif aux ressources et aux services disponibles sur un compte AWS. Consultez Terminologie dans le Guide de l'utilisateur de configuration d'AWS.	✓	✓	✓
ARN	Amazon Resource Name, un identifiant unique d'une ressource AWS. Les formats ARN spécifiques dépendent du type de ressource. Pour les formats d'ARN utilisés par les ressources IVS, consultez Référence de l'autorisation de service.	✓	✓	✓
Proportions	Décrit le rapport entre la largeur du cadre et la hauteur du cadre. Par exemple, 16:9 représente les	✓	✓	

Terme	Description	LL	RT	Chat
	proportions qui correspondent à la résolution Full HD ou 1080p.			
Mode audio	Configuration audio prédéfinie ou personnalisée optimisée pour différents types d'utilisateurs d'appareils mobiles et pour l'équipement qu'ils utilisent. Consultez SDK de diffusion IVS : Modes audio mobiles (Streaming en temps réel).		✓	
AVC, H.264, MPEG-4 partie 10	Le codage vidéo avancé, également appelé H.264 ou MPEG-4 partie 10, est une norme de compression vidéo pour la compression vidéo numérique avec perte.	✓	✓	
Remplacement d'arrière-plan	Type de filtre de caméra permettant aux créateurs de flux en direct de modifier leur arrière-plan. Consultez Remplacement d'arrière-plan dans SDK de diffusion IVS : filtres de caméra tiers (Streaming en temps réel).		✓	
Débit binaire	Une métrique de streaming pour le nombre de bits transmis ou reçus par seconde.	✓	✓	
Diffusion, diffuseur	Autres termes pour flux et streamer.	✓		
Mise en mémoire tampon	Une condition qui se produit lorsque le périphérique de lecture n'est pas en mesure de télécharger le contenu avant qu'il ne soit censé être lu. La mise en mémoire tampon peut se manifester de différentes manières : le contenu peut s'arrêter et se relancer de manière aléatoire (également appelé saut d'image), le contenu peut s'arrêter pendant de longues périodes (également connu sous le nom de blocage) ou le lecteur IVS peut suspendre la lecture.	✓	✓	

Terme	Description	LL	RT	Chat
Listes de lecture par plage d'octets	Une liste de lecture plus détaillée que la liste de lecture HLS standard. La liste de lecture HLS standard est composée de fichiers multimédias de 10 secondes. Avec une liste de lecture par plage d'octets, la durée du segment est la même que l'intervalle d'images clés configuré pour le flux. La liste de lecture par plage d'octets n'est disponible que pour les diffusions enregistrées automatiquement dans un compartiment S3. Elle est créée en complément de la liste de lecture HLS. Consultez Liste de lecture par plage d'octets dans Enregistrement automatique vers Amazon S3 (streaming à faible latence).	✓		
CBR	Débit binaire constant, une méthode de contrôle du débit pour les encodeurs qui maintient un débit binaire constant pendant toute la durée de lecture d'une vidéo, indépendamment de ce qui se passe pendant la diffusion. Les périodes d'accalmie peuvent être atténuées pour atteindre le débit souhaité, et les pics peuvent être quantifiés en ajustant la qualité du codage pour qu'elle corresponde au débit binaire cible. Nous recommandons vivement d'utiliser le débit constant (CBR) au lieu du débit variable (VBR).	✓	✓	
CDN	Réseau de diffusion de contenu ou réseau de distribution de contenu, une solution distribuée géographiquement qui optimise la diffusion de contenu tel que les vidéos en streaming en le rapprochant de l'emplacement des utilisateurs.	✓		

Terme	Description	LL	RT	Chat
Canal	Ressource IVS qui stocke la configuration pour le streaming, y compris un serveur d'ingestion, une clé de flux, une URL de lecture et des options d'enregistrement. Les streamers utilisent la clé de flux associée à un canal pour démarrer une diffusion. Tous les événements et métriques générés lors d'une diffusion sont associés à une ressource de canal.	✓		
Type de canal	Détermine la résolution et la fréquence d'images autorisées pour le canal. Consultez la section Types de canaux (français non garanti) dans Référence de l'API de diffusion à faible latence d'IVS.	✓		
Journalisation du chat	Option avancée qui peut être activée en associant une configuration de journalisation à une salle de chat.			✓
Salle de chat	Ressource IVS qui stocke la configuration d'une session de chat, y compris des fonctionnalités facultatives telles que Gestionnaire de révision des messages et Journalisation du chat. Consultez Étape 2 : créer une salle de chat dans Mise en route avec le chat IVS.			✓
Montage côté client	Utilise un appareil hôte pour mixer les flux audio et vidéo des participants à l'étape, puis les envoie sous forme de flux composite vers un canal IVS. Cela permet de mieux contrôler l'aspect de la composition au prix d'une utilisation plus importante des ressources des clients et d'un risque plus élevé qu'un problème lié à l'étape ou à l'hôte ait un impact sur les utilisateurs. Consultez également Montage côté serveur.	✓	✓	

Terme	Description	LL	RT	Chat
CloudFront	Service CDN fourni par Amazon.	✓		
CloudTrail	Service AWS permettant de collecter, de surveille r, d'analyser et de retenir les événements et les activités du compte provenant d'AWS et de sources externes. Consultez la section Journalisation des appels d'API IVS avec AWS CloudTrail.	✓	✓	✓
CloudWatch	Service AWS permettant de surveiller les applicati ons, de répondre aux changements de performan ces, d'optimiser l'utilisation des ressources et de fournir des informations sur l'état opérationnel. Vous pouvez l'utiliser CloudWatch pour surveiller les métriques IVS ; voir Surveillance du streaming IVS en temps réel et Surveillance du streaming IVS à faible latence.	✓	✓	✓
Montage	Processus consistant à combiner des flux audio et vidéo provenant de plusieurs sources en un seul flux.	✓	✓	
Pipeline de montage	Séquence d'étapes de traitement requise pour combiner plusieurs flux et encoder le flux résultant.	✓	✓	
Compression	Codage des informations en utilisant moins de bits que la représentation d'origine. Toute compressi on particulière est soit sans perte, soit avec perte. La compression sans perte réduit le nombre de bits en identifiant et en éliminant la redondance statistique. Aucune information n'est perdue lors de la compression sans perte. La compression avec perte réduit le nombre de bits en supprimant les informations inutiles ou de moindre importance.	✓	✓	

Terme	Description	LL	RT	Chat
Plan de contrôle	Stocke des informations sur les ressources IVS telles que les canaux, les étapes ou les salles de chat et fournit des interfaces pour créer et gérer ces ressources. Il est régional (basé sur les Régions AWS).	✓	✓	✓
CORS	Le partage des ressources cross-origin (CORS), une fonctionnalité AWS qui permet aux applications Web clientes chargées dans un domaine particuli er d'interagir avec les ressources, telles que des compartiments S3 d'un autre domaine. L'accès peut être configuré en fonction des en-têtes, des méthodes HTTP et des domaines d'origine. Consultez Utilisation du partage des ressources entre origines multiples (CORS) - Amazon Simple Storage Service dans le Guide de l'utilisateur Amazon Simple Storage Service.	✓		
Source d'image personnalisée	Interface fournie par le SDK de diffusion IVS qui permet à une application de fournir sa propre entrée d'image au lieu de se limiter aux caméras prédéfinies.	✓	✓	
Plan de données	L'infrastructure qui transporte les données de l'entrée jusqu'à la sortie. Elle fonctionne sur la base de la configuration gérée dans le plan de contrôle et n'est pas limitée à une Région AWS.	✓	✓	✓
Encodeur, encodage	Le processus de conversion de contenu vidéo et audio en un format numérique, adapté au streaming. L'encodage peut être matériel ou logiciel.	✓	✓	

Terme	Description	LL	RT	Chat
Événement	Une notification automatique publiée par IVS au service de AmazonEventBridge surveillance. Un événement représente une modification de l'état ou de l'intégrité d'une ressource de streaming telle qu'une étape ou un pipeline de montage. Consultez les sections Utilisation d'Amazon EventBridge avec le streaming à faible latence IVS et Utilisation d'Amazon EventBridge avec le streaming en temps réel IVS.	✓	✓	✓
FFmpeg	Un projet logiciel gratuit et open source composé d'une suite de bibliothèques et de programmes pour gérer des fichiers et des flux vidéo et audio. FFmpeg propose une solution multiplateforme pour enregistrer, convertir et diffuser du contenu audio et vidéo.	✓		
Flux fragmenté	Créé lorsqu'une diffusion se déconnecte puis se reconnecte durant l'intervalle spécifié dans la configuration d'enregistrement du canal. Les flux multiples qui en résultent sont considérés comme une diffusion unique et sont fusionnés en un flux enregistré unique. Consultez Fusionner des flux fragmentés dans Enregistrement automatique vers Amazon S3 (streaming à faible latence).	✓		
Fréquence de trames	Une métrique de streaming pour le nombre de trames vidéo transmises ou reçues par seconde.	✓	✓	
HLS	HTTP Live Streaming (HLS), un protocole de communication de streaming à débit binaire adaptatif basé sur le protocole HTTP et utilisé pour transmettre des flux IVS aux utilisateurs.	✓		

Terme	Description	LL	RT	Chat
Liste de lecture HLS	Liste des segments multimédias qui constitue nt un flux. Les listes de lecture HLS standard sont composées de fichiers multimédias de 10 secondes. HLS prend également en charge des listes de lecture plus détaillées par plage d'octets.	✓		
Host (Hôte)	Un participant à un événement en temps réel qui envoie de la vidéo et/ou du son à la scène.		✓	
IAM	Identity and Access Management, un service AWS qui permet aux utilisateurs de gérer en toute sécurité les identités et l'accès aux services et aux ressources AWS, y compris IVS.	✓	✓	✓
Ingestion	Processus IVS pour recevoir des flux vidéo d'un hôte ou d'un diffuseur à des fins de traitement ou de diffusion aux utilisateurs ou à d'autres participa nts.	✓	✓	
Serveur d'ingestion	Reçoit les flux vidéo et les transmet à un système de transcodage, où les flux sont transmutés ou transcodés en HLS pour être transmis aux spectateurs. Les serveurs d'ingestion sont des composants IVS spécifiques qui reçoivent des flux pour les canaux, ainsi qu'un protocole d'ingestion (RTMP, RTMPS). Consultez les informations sur la création d'un canal dans Mise en route avec le streaming à faible latence IVS.		✓	

Terme	Description	LL	RT	Chat
Vidéo entrelacée	Transmet et affiche uniquement les lignes paires ou impaires des trames suivantes afin de créer une impression de doublement de la fréquence de trames sans consommer de bande passante supplémentaire. Nous vous déconseillons d'utiliser la vidéo entrelacée pour des raisons liées à la qualité de la vidéo.	✓	✓	
JSON	JavaScript Notation d'objet, format de fichier standard ouvert qui utilise du texte lisible par l'homme pour transmettre des objets de données composés de paires attribut-valeur et de types de données de tableau ou d'autres valeurs sérialisables.	✓	✓	✓
Image clé, image delta, intervalle d'image clé	L'image clé (également appelée image intra-codée ou i-Frame) est une image complète de l'image d'une vidéo. Les images suivantes, les images delta (également appelées images prédites ou p-Frames), contiennent uniquement les informations modifiées. Les images-clés apparaîtront plusieurs fois dans un flux, en fonction de l'intervalle d'images clés défini dans l'encodeur.	✓	✓	
Lambda	Un service AWS permettant d'exécuter du code (appelé fonctions Lambda) sans allouer d'infrastructure de serveur. Les fonctions Lambda peuvent être exécutées en réponse à des événements et à des demandes d'invocation, ou selon un calendrier. Par exemple, le chat IVS utilise les fonctions Lambda pour permettre la révision des messages dans une salle de chat.	✓	✓	✓

Terme	Description	LL	RT	Chat
Latence, glass-to-glass latence	Un retard dans le transfert de données. IVS définit les plages de latence comme suit : • Faible latence : moins de 3 secondes • Latence en temps réel : moins de 300 ms La lass-to-glass latence G fait référence au délai entre le moment où une caméra capture un flux en direct et le moment où le flux apparaît sur l'écran du spectateur.	✓	✓	
Codage en couches avec Simulcast.	Permet le codage et la publication simultanés de plusieurs flux vidéo avec différents niveaux de qualité. Consultez Streaming adaptatif : encodage en couches avec Simulcast dans Optimisations du streaming en temps réel.		✓	
Gestionnaire de révision des messages	Permet aux clients du chat IVS de consulter/filtrer automatiquement les messages de chat des utilisateurs avant qu'ils ne soient envoyés dans la salle de chat. Il est activé en associant une fonction Lambda à une salle de chat. Consultez Creating a Lambda Function dans Chat Message Review Handler.			✓

Terme	Description	LL	RT	Chat
Mixeur	Une fonctionnalité des SDK de diffusion mobile IVS qui prend plusieurs sources audio et vidéo pour et générer une seule sortie. Elle prend en charge la gestion des éléments vidéo et audio à l'écran représentant des sources telles que des caméras, des microphones, des captures d'écran, ainsi que de l'audio et de la vidéo générés par l'applica tion. La sortie peut ensuite être transmise à IVS. Consultez Configuration d'une séance de diffusion pour le mixage dans SDK de diffusion IVS : guide de mixage (Streaming à faible latence).	✓		
Streaming multi-hôt es	Combine les flux provenant de plusieurs hôtes en un seul flux. Cela peut être accompli en utilisant un montage côté client ou côté serveur. Le streaming multi-hôtes permet des scénarios tels que l'invitation de spectateurs sur scène pour des questions-réponses, les compétitions entre hôtes, le chat vidéo et les conversations entre hôtes devant un large public.		✓	
Liste de lecture multivariante	Un index de tous les flux de variantes disponibles pour une diffusion.	✓		
OAC	Origin Access Control, un mécanisme permettant de restreindre l'accès à un compartiment S3, afin que le contenu tel qu'un flux enregistré ne puisse être diffusé que via un CloudFrontCDN.	✓		

Terme	Description	LL	RT	Chat
OBS	Open Broadcaster Software, logiciel gratuit et open source pour l'enregistrement et la diffusion en direct de vidéos. OBS propose une alternative (au SDK de diffusion IVS) pour la publication sur ordinateur. Les streamers plus sophistiqués qui connaissent bien OBS peuvent le préférer en raison de ses fonctionnalités de production avancées, telles que les transitions de scène, le mixage audio et la superposition des graphiques.	✓	✓	
Participant	Un utilisateur en temps réel connecté à une scène en tant qu'hôte ou utilisateur.		✓	
Jeton de participant	Authentifie un participant à un événement en temps réel lorsqu'il rejoint une scène. Un jeton de participant contrôle également si un participant peut envoyer une vidéo à la scène.		✓	
Jeton de lecture, paire de clés de lecture	Un mécanisme d'autorisation qui permet aux clients de restreindre la lecture de vidéos sur les canaux privés. Les jetons de lecture sont générés à partir d'une paire de clés de lecture. Une paire de clés de lecture est la paire de clés publique-privée utilisée pour signer et valider le jeton d'autorisation de l'utilisateur pour la lecture. Consultez Créer ou importer une clé de lecture dans Configurer des canaux privés et voir les points de terminaison de la paire de clés de lecture dans Référence d'API IVS à faible latence.	✓		

Terme	Description	LL	RT	Chat
URL de lecture	Identifie l'adresse utilisée par l'utilisateur pour lancer la lecture d'un canal spécifique. Cette adresse peut être utilisée partout dans le monde. IVS sélectionne automatiquement le meilleur emplacement sur le réseau mondial de streaming de contenu IVS afin de diffuser la vidéo à chaque utilisateur. Consultez les informations sur la création d'un canal dans Mise en route avec le streaming à faible latence IVS.	✓		
Canal privé	Permet aux clients de restreindre l'accès à leurs flux à l'aide d'un mécanisme d'autorisation basé sur des jetons de lecture. Voir Flux de travail pour les canaux privés dans Configurer des canaux privés.	✓		
Vidéo progressive	Transmet et affiche toutes les lignes de chaque image en séquence. Nous recommandons d'utilise r la vidéo progressive à toutes les étapes d'une diffusion.	✓	✓	
Quotas	Le nombre maximal de ressources ou d'opérations de service IVS pour votre compte AWS. Autrement dit, ces limites s'entendent par compte AWS, sauf indication contraire. Tous les quotas sont appliqués par région. Consultez Amazon Interactive Video Service endpoints and quotas dans Guide de référence général AWS.	✓	✓	✓

Terme	Description	LL	RT	Chat
Régions	Elles permettent d'accéder aux services AWS qui résident physiquement dans une région géographique spécifique. Les régions fournissent une tolérance aux pannes, une stabilité et une résilience, et peuvent également réduire la latence. Les régions vous permettent de créer des ressources redondantes qui restent disponibles et qui ne sont pas affectées par une panne régionale. La plupart des demandes de service AWS sont associées à une région géographique particulière. Les ressources que vous créez dans une région n'existent pas dans une autre région, sauf si vous utilisez explicitement une fonction de réplication offerte par un service AWS. Par exemple, Amazon S3 prend en charge la réplication entre régions. Certains services, tels qu'IAM, n'ont pas de ressources interrégionales.	✓	✓	✓
Résolution	Décrit le nombre de pixels d'une seule image vidéo. Par exemple, Full HD ou 1080p définit une image de 1920 x 1080 pixels.	✓	✓	
Utilisateur root	Propriétaire d'un compte AWS. L'utilisateur root a un accès total à tous les services et ressources AWS du compte AWS.	✓	✓	✓
RTMP, RTMPS	Real-Time Messaging Protocol, une norme du secteur pour la transmission d'audio, de vidéos et de données sur un réseau. RTMPS est la version sécurisée de RTMP, qui s'exécute sur une connexion de protocole TLS/SSL (Transport Layer Security).	✓	✓	

Terme	Description	LL	RT	Chat
Compartiment S3	Un ensemble d'objets stockés dans Amazon S3. De nombreuses politiques, y compris l'accès et la réplication, sont définies au niveau du compartiment et s'appliquent à tous les objets du compartiment. Par exemple, une diffusion IVS est stockée en tant qu'objets multiples dans un compartiment S3.	✓		
Kit SDK	Kit de développement logiciel, une collection de bibliothèques pour les développeurs qui créent des applications avec IVS.	✓	✓	✓
Segmentation des selfies	Permet de remplacer l'arrière-plan dans un flux en direct, à l'aide d'une solution propre au client qui accepte une image de caméra en entrée et qui renvoie un masque fournissant un score de confiance pour chaque pixel de l'image, indiquant s'il se trouve au premier plan ou en arrière-plan. Consultez Remplacement d'arrière-plan dans SDK de diffusion IVS : filtres de caméra tiers (Streaming en temps réel).		✓	
Gestion des versions sémantique	Un format de version au format Major.Minor.Patch. Les corrections de bogues n'affectant pas l'API incrémentent la version du correctif, les ajouts/modifications d'API rétrocompatibles incrémentent la version mineure et les modifications d'API non compatibles avec les versions antérieures incrémentent la version majeure.	✓	✓	✓

Terme	Description	LL	RT	Chat
Montage côté serveur	Utilise un serveur IVS pour mixer le son et la vidéo des participants à la scène, puis envoie cette vidéo mixée à un canal IVS pour atteindre un public plus large ou la stocker dans un compartiment S3. Le montage côté serveur réduit la charge client, améliore la résilience de la diffusion et permet une utilisation plus efficace de la bande passante. Consultez également Montage côté client.		✓	
Quotas de service	Service AWS qui vous permet de gérer vos quotas pour de nombreux services AWS à partir d'un seul emplacement. En plus de la recherche des valeurs des quotas, vous pouvez également demander à augmenter un quota à partir de la console Service Quotas.	✓	✓	✓
Rôle lié à un service	Un type unique de rôle IAM directement lié à un service AWS. Les rôles liés à des services sont automatiquement créés par IVS et ils incluent toutes les autorisations requises par le service pour appeler d'autres services AWS en votre nom, par exemple pour accéder à un compartiment S3. Consultez Using Service-Linked Roles for IVS dans IVS Security.	✓		
Étape	Une ressource IVS qui représente un espace virtuel où les participants à un événement en temps réel peuvent échanger des vidéos en temps réel. Consultez Créer une étape dans Mise en route avec le streaming en temps réel IVS.		✓	

Terme	Description	LL	RT	Chat
Session d'étape	Elle commence lorsque le premier participant rejoint une étape et se termine quelques minutes après que le dernier participant cesse d'être diffusé sur l'étape. Une étape de longue durée peut comporter plusieurs sessions au cours de sa durée de vie.		✓	
Flux	Données représentant un contenu vidéo ou audio envoyé en continu d'une source vers une destination.	✓	✓	
Clé de flux	Identifiant attribué par IVS lors de la création d'un canal et utilisé pour autoriser le streaming sur le canal. Traitez la clé de flux comme un secret, car elle permet à n'importe qui de diffuser sur le canal. Consultez Mise en route avec le streaming à faible latence IVS.	✓		
Pénurie de flux	Retard ou arrêt de la diffusion du flux vers IVS. Cela se produit lorsqu'IVS ne reçoit pas le nombre de bits que le dispositif d'encodage avait annoncé envoyer sur une certaine période. La survenue d'une pénurie de flux entraîne un événement de pénurie de flux. Pour le spectateur, une pénurie de flux peut se traduire par un retard, une mise en mémoire tampon ou un blocage d'une vidéo. La pénurie de flux peut être brève (moins de 5 secondes) ou longue (plusieurs minutes), selon la situation spécifique qui l'a provoquée. Consultez Qu'est-ce que la pénurie de flux ? dans Questions fréquentes sur le dépannage.	✓	✓	
Streamer	Personne ou appareil envoyant un flux vidéo ou audio à IVS.	✓	✓	

Terme	Description	LL	RT	Chat
Subscriber	Un participant à un événement en temps réel qui reçoit de la vidéo et/ou du son de l'hôte. Consultez Qu'est-ce qu'IVS Real-Time Streaming ?		✓	
Balise	Une balise de métadonnées est une étiquette que vous affectez à une ressource AWS. Les balises peuvent vous aider à identifier et à organiser vos ressources AWS. Sur la page d'accueil de la documentation IVS, consultez « Balisage » dans n'importe quelle documentation de l'API IVS (pour le streaming en temps réel, le streaming à faible latence ou le chat).	✓	✓	✓
Filtres de caméras tiers	Composants logiciels qui peuvent être intégrés au SDK de diffusion IVS pour permettre à une application de traiter des images avant de les transmettre au SDK de diffusion en tant que source d'image personnalisée. Un filtre de caméra tiers peut traiter les images provenant de la caméra, appliquer un effet de filtre, etc.	✓	✓	
Miniature	Image de taille réduite provenant d'un flux. Par défaut, les miniatures sont générées toutes les 60 secondes, mais un intervalle plus court peut être configuré. La résolution des miniatures dépend du type de canal. Consultez Enregistrement des contenus dans Enregistrement automatique vers Amazon S3 (streaming à faible latence).	✓		

Terme	Description	LL	RT	Chat
Métadonnées temporisées	Métadonnées liées à des horodatages spécifiques au sein d'un flux. Elles peuvent être ajoutées par programmation à l'aide de l'API IVS et sont associées à des images spécifiques. Cela garantit que tous les utilisateurs reçoivent les métadonnées au même point par rapport au flux. Les métadonnées temporisées peuvent être utilisées pour déclencher des actions sur le client, telles que la mise à jour des statistiques de l'équipe lors d'un événement sportif. Consultez Intégration de métadonnées dans un flux vidéo.	✓		
Transcodage	Convertit de la vidéo et de l'audio d'un format à un autre. Un flux entrant peut être transcodé dans un format différent à plusieurs débits et résolutions afin de prendre en charge une gamme de périphériques de lecture et de conditions réseau.	✓	✓	
Transmuxage	Un simple reconditionnement d'un flux ingéré vers IVS, sans réencodage du flux vidéo. « Transmux » est un raccourci pour le multiplexage transcodé , un processus qui change le format d'un fichier audio et/ou vidéo tout en conservant une partie ou la totalité des flux d'origine. Cela convertit vers un format de conteneur différent sans modifier le contenu du fichier. Il se distingue du transcodage.	✓	✓	

Terme	Description	LL	RT	Chat
Flux de variante	Ensemble d'encodages d'une même diffusion selon plusieurs niveaux de qualité distincts. Chaque flux de variante de flux est codé sous forme de liste de lecture HLS distincte. Un index des flux de variantes disponibles est appelé liste de lecture multivariante. Une fois que le lecteur IVS a reçu une liste de lecture multivariante d'IVS, il peut alors choisir entre les différents flux de variantes pendant la lecture, en passant de l'un à l'autre de manière transparente en fonction des conditions du réseau.	✓		
VBR	Débit binaire variable, méthode de contrôle du débit pour les encodeurs qui utilise un débit binaire dynamique qui change tout au long de la lecture, en fonction du niveau de détail requis. Nous vous déconseillons vivement d'utiliser le VBR pour des raisons de qualité de la vidéo. Utilisez plutôt le CBR.	✓	✓	

Terme	Description	LL	RT	Chat
Vue	Session de visionnage unique qui est en train de télécharger ou de lire activement des vidéos. Les vues constituent la base du quota de vues simultanées. Une vue démarre lorsqu'une session de visualisation lance la lecture vidéo. Une vue se termine lorsqu'une session de visualisation arrête la lecture vidéo. La lecture est le seul indicateur de l'audience ; les heuristiques d'engagement telles que les niveaux audio, la mise au point de l'onglet du navigateur et la qualité de la vidéo ne sont pas prises en compte. Lorsque vous comptez les vues, IVS ne tient pas compte de la légitimité des utilisateurs individuels et ne tente pas de dédupliquer les spectateurs localisés, par exemple dans le cas de plusieurs lecteurs vidéo sur une seule machine. Consultez Autres quotas dans Service Quotas (Streaming à faible latence).	✓		
Lecteur	Personne recevant un flux d'IVS.	✓		

Terme	Description	LL	RT	Chat
WebRTC	Web Real-Time Communication, un projet open source proposant une communication en temps réel aux navigateurs Web et aux applications mobiles. Il permet aux communications audio et vidéo de fonctionner au sein des pages Web en permettant une peer-to-peer communication directe, éliminant ainsi le besoin d'installer des plugins ou de télécharger des applications natives. Les technologies sous-jacentes au WebRTC sont mises en œuvre sous la forme d'un standard Web ouvert et sont disponibles sous forme d'API JavaScript standard dans tous les principaux navigateurs ou sous forme de bibliothèques pour les clients natifs, tels qu'Android et iOS.	✓	✓	

Terme	Description	LL	RT	Chat
FOUET	Protocole d'ingestion WebRTC-HTTP, protocole basé sur HTTP qui permet l'ingestion de contenu via WebRTC dans des services de streaming et/ou des CDN. WHIP est un projet de l'IETF développé pour normaliser l'ingestion du WebRTC. WHIP assure la compatibilité avec des logiciels tels que OBS, offrant une alternative (au SDK de diffusion IVS) pour la publication assistée par ordinateur. Les streamers plus sophistiqués qui connaissent bien OBS peuvent le préférer pour ses fonctionnalités de production avancées, telles que les transitions de scène, le mixage audio et les graphismes superposés WHIP est également utile dans les situations où l'utilisation du SDK de diffusion IVS n'est pas faisable ou préférable. Par exemple, dans les configurations impliquant des encodeurs matériels , le SDK de diffusion IVS peut ne pas être une option. Toutefois, si l'encodeur prend en charge le protocole WHIP, vous pouvez toujours publier directement depuis l'encodeur vers IVS. Voir Support OBS et WHIP.		✓	
WSS	WebSocket Secure, protocole permettant d'établir une WebSockets connexion TLS cryptée. Il sert à se connecter aux points de terminaison de chat IVS. Consultez Étape 4 : envoyer et recevoir votre premier message dans Mise en route avec le chat IVS.			✓

Historique du document (Streaming en temps réel)

Modifications apportées au guide utilisateur du streaming en temps réel

Modification	Description	Date
Support OBS et WHIP	Ajout d'une nouvelle page. Ce document explique comment utiliser des encodeurs compatibles WHIP tels que OBS pour publier sur le streaming en temps réel IVS. Le WHIP (WebRTC-HTTP Ingestion Protocol) est un projet de l'IETF développé pour normaliser l'ingestion du WebRTC.	6 février 2024
SDK de diffusion : Android 1.14.1, iOS 1.14.1, Web 1.8.0	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans les guides du SDK de real-time-streaming diffusion : Android, iOS et Web. Sur la page de destination de la documentation Amazon IVS, mise à jour des liens de référence du kit SDK du lecteur pour vous rediriger vers la nouvelle version. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	1 février 2024

	Pour le guide Android, nous avons ajouté un nouveau problème connu (taille de vidéo inférieure à 176 x 176). Pour le guide Web, nous avons ajouté un nouveau problème connu. La solution consiste à limiter la résolution vidéo à 720p lors de l'invocation de ou. `getUserMedia` `getDisplayMedia` Dans la section Optimisations du streaming en temps réel, nous avons mis à jour la configuration du codage en couches avec Simulcast ; cette option est désormais désactivée par défaut.	
SDK de diffusion : Android 1.13.4, iOS 1.13.4, Web 1.7.0	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans les guides du SDK de real-time-streaming diffusion : Android, iOS et Web. Sur la page de destination de la documentation Amazon IVS, mise à jour des liens de référence du kit SDK du lecteur pour vous rediriger vers la nouvelle version. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	3 janvier 2024

Glossaire IVS	Le glossaire a été étendu pour couvrir les termes IVS en temps réel, faible latence et chat.	20 décembre 2023
Stage Health : nouvelles CloudWatch mesures	La métrique PacketLoss (Stage) a été renommée DownloadPacketLoss (Stage) et a publié des CloudWatch métriques supplémentaires pour le streaming en temps réel IVS : • DownloadPacketLoss (Étape, participant) • DroppedFrames (Étape, participant) • SubscribeBitrate (Étape, participant,MediaType) Consultez Surveillance du streaming en temps réel IVS.	7 décembre 2023
Politiques gérées IAM	Ajout de deux politiques gérées, IVS ReadOnlyAccess et FullAccess IVS. Consultez : • La nouvelle section sur les politiques gérées pour Amazon IVS se trouve sur la page Sécurité. • Modifications apportées à Étape 3 : configurer des autorisations IAM dans Mise en route avec le streaming à faible latence IVS.	5 décembre 2023

SDK de diffusion : Android 1.13.2 et iOS 1.13.2	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans les guides du SDK de real-time-streaming diffusion : Android et iOS. Sur la page de destination de la documentation Amazon IVS, mise à jour des liens de référence du kit SDK du lecteur pour vous rediriger vers la nouvelle version. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	4 décembre 2023
SDK de diffusion : Android 1.13.1	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans le guide du SDK de real-time-streaming diffusion : Android. Sur la page de destination de la documentation Amazon IVS, mise à jour des liens de référence du kit SDK du lecteur pour vous rediriger vers la nouvelle version. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	21 novembre 2023
Service Quotas	La « résolution de diffusion des participants » est passée de 1080p à 720p.	18 novembre 2023

SDK de diffusion : Android 1.13.0 et iOS 1.13.0	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans les guides du SDK de real-time-streaming diffusion : Android et iOS. Sur la page de destination de la documentation Amazon IVS, mise à jour des liens de référence du kit SDK du lecteur pour vous rediriger vers la nouvelle version. Voir également les Notes de mise à jour d'Amazon IVS pour cette version. Nous avons également apporté diverses mises à jour aux Optimisations du streaming. Entre autres, la fonctionnalité « Streaming adaptatif : encodage en couches avec diffusion simultanée » nécessite désormais une inscription explicite et n'est prise en charge que dans les versions récentes du SDK.	17 novembre 2023

Enregistrement composite	Modifications suivantes effectuées : • Ajout d'une page Enregistrement composite pour cette nouvelle fonctionnalité. • Mise à jour de Premiers pas avec le streaming en temps réel IVS avec intégration de points de terminaison S3 dans la politique au sein de « Configurer les autorisations IAM ». • Mise à jour des Service Quotas avec les taux d'appels pour les nouveaux points de terminaison.	16 novembre 2023

Montage côté serveur (SSC)	Le montage côté serveur IVS permet aux clients de décharger le montage et la diffusion d'une scène IVS vers un service géré par IVS. Les diffusions SSC et RTMP vers un canal sont appelées via les points de terminaison du plan de contrôle IVS situés dans la région d'origine de la scène. Consultez : • Démarrer : nous avons ajouté des points de terminaison SSC à la politique dans « Configurer les autorisations IAM ». • Utilisation d'Amazon EventBridge avec IVS — Nous avons ajouté de nouvelles statistiques. • Montage côté serveur : ce nouveau document inclut un aperçu et des instructions de configuration. • Service Quotas : nous avons ajouté de nouvelles limites de débit d'appels et d'autres quotas. Voir aussi : • liste des changements concernant les Modifications des références de l'API	16 novembre 2023

	de streaming en temps réel IVS. • Liste des changements concernant l'Historique du document (Streaming à faible latence).	
SDK de diffusion IVS	Dans la présentation du SDK de diffusion, nous avons mis à jour Configuration requise > Plateformes natives pour préciser quelles versions du SDK sont prises en charge, et nous avons ajouté « Navigateurs mobiles (iOS et Android) ». Dans le Guide Web de diffusion, nous avons ajouté les « limites du Web mobile ».	9 novembre 2023
SDK de diffusion IVS	Nous avons ajouté une nouvelle page sur les filtres de caméra tiers.	9 novembre 2023
Mise en route avec la diffusion en temps réel d'IVS	Nous avons mis à jour les procédures dans Configuration des autorisations IAM.	20 octobre 2023
Surveillance du streaming en temps réel	Dans CloudWatch Metrics : IVS Real-Time Streaming , nous avons ajouté des exemples de valeurs pour les dimensions.	17 octobre 2023

Kit SDK de diffusion : guide pour le Web	Nous avons apporté plusieurs modifications pour Surveiller l'état muet du contenu multimédia des participants distants.	17 octobre 2023
Kit SDK de diffusion : Web 1.6.0	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans le guide du SDK de real-time-streaming diffusion : Web. La page de destination de la documentation Amazon IVS renvoie à la version actuelle des références du kit SDK de diffusion. Voir également les Notes de mise à jour d'Amazon IVS pour cette version. Dans le Guide Web, dans la section « Retrieve a MediaStream from a device », nous avons également supprimé les deux `max` lignes ; la meilleure pratique consiste à ne spécifier que`ideal`. Dans Optimisations du streaming en temps réel, nous avons ajouté une nouvelle section intitulée Optimisation du débit audio et du support stéréo.	16 octobre 2023

Stage Health : nouvelles CloudWatch mesures	Statistiques publiées CloudWatch pour le streaming en temps réel IVS. Consultez Surveillance du streaming en temps réel IVS.	12 octobre 2023
Kit SDK de diffusion : Android 1.12.1	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans le guide du SDK de real-time-streaming diffusion : Android. Nous avons également ajouté une nouvelle section, Utilisation de microphones Bluetooth. La page de destination de la documentation Amazon IVS renvoie à la version actuelle des références du kit SDK de diffusion. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	12 octobre 2023

Kit SDK de diffusion : Web 1.5.2	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans le guide du SDK de real-time-streaming diffusion : Web. La page de destination de la documentation Amazon IVS renvoie à la version actuelle des références du kit SDK de diffusion. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	14 septembre 2023
Mise en route avec la diffusion en temps réel d'IVS	Dans Android > Installer le kit SDK de diffusion, la liaison de données a été ajoutée.	12 septembre 2023
Gestion des erreurs du kit SDK de diffusion	Ajout de sections « Gestion des erreurs » aux guides du kit SDK de diffusion : Web, Android et iOS.	12 septembre 2023
Mise en route avec la diffusion en temps réel d'IVS	Ajout d'une note importante dans Distribuer les jetons des participants sur le fait que la création de la fonctionnalité ne doit pas se baser sur le format actuel des jetons.	1er septembre 2023
Mise en route avec la diffusion en temps réel d'IVS	Mise à jour de l'ensemble des autorisations dans Configurer les autorisations IAM.	31 août 2023

SDK de diffusion : Web 1.5.1, Android 1.12.0 et iOS 1.12.0	Numéro de version et liens d'artefacts mis à jour pour la nouvelle version, dans les guides du SDK de real-time -streaming diffusion : Web, Android et iOS. Sur la page de destination de la documentation Amazon IVS, mise à jour des liens de référence du kit SDK du lecteur pour vous rediriger vers la nouvelle version. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	23 août 2023

Lancement du streaming en temps réel	Des modifications majeures de la documentation accompagnent cette version. Nous avons renommé la documentation précédente en « Streaming à faible latence IVS » et publié une nouvelle documentation relative au « Streaming en temps réel IVS ». La page d'accueil de la documentation IVS comporte désormais des sections distinctes pour le streaming en temps réel et le streaming à faible latence. Chaque section possède son propre guide de l'utilisateur et sa propre référence d'API. Pour les autres modifications apportées à la documentation, consultez la section Historique du document (Streaming à faible latence).	7 août 2023

SDK de diffusion : Web 1.5.0, Android 1.11.0 et iOS 1.11.0	Mise à jour du numéro de version et des liens d'artefact pour la nouvelle version, dans les guides du kit SDK de diffusion : Web, Android et iOS. Sur la page de destination de la documentation Amazon IVS, mise à jour des liens de référence du kit SDK du lecteur pour vous rediriger vers la nouvelle version. Voir également les Notes de mise à jour d'Amazon IVS pour cette version.	7 août 2023

Modifications de la référence de l'API de streaming en temps réel IVS

Modifications d'API	Description	Date
Enregistrement composite	Nous avons ajouté 4 StorageConfiguration points de terminaison et 7 objets (DestinationDetail,, S3 RecordingConfigurationDestinationConfiguration, S3Detail, S3,StorageConfiguration, StorageConfiguration). StorageConfigurationSummary Nous avons modifié 3 objets (Composition, Destination, DestinationConfiguration). Cela affecte la GetComposition réponse, la StartComposition demande et la réponse.	16 novembre 2023
Montage côté serveur	Nous avons ajouté 8 compositions et EncoderConfiguration points de terminaison et 11 objets	16 novembre 2023

Modifications d'API	Description	Date
	(Composition ChannelDestinationConfiguration CompositionSummary, Destination DestinationConfiguration, DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration LayoutConfiguration, et Vidéo).	
État de l'étape : nouvelles données sur les participants	Six champs ont été ajoutés à l'objet Participant : `browserName` , `browserVersion` , `ispName`, `osName`, `osVersion` et `sdkVersion` . Cela affecte la GetParticipant réponse.	12 octobre 2023
Jeton de participant	Ajout d'une note importante sur le fait que la création de la fonctionnalité ne doit pas se baser sur le format actuel des jetons.	1er septembre 202
Lancement du streaming en temps réel IVS	Des modifications majeures de la documentation accompagnent cette version. Nous avons renommé la documentation précédente en « Streaming à faible latence IVS » et publié une nouvelle documentation relative au « Streaming en temps réel IVS ». La page d'accueil de la documentation IVS comporte désormais des sections distinctes pour le streaming en temps réel et le streaming à faible latence. Chaque section possède son propre guide de l'utilisateur et sa propre référence d'API. La Référence de l'API de streaming en temps réel IVS fait partie de la documentation relative au streaming en temps réel IVS. Auparavant, elle s'intitulait Référence de l'API IVS Scène. Son historique est décrit dans la section Historique du document (Streaming à faible latence).	7 août 2023

Notes de publication (Streaming en temps réel)

6 février 2024

Support OBS et WHIP

IVS peut être utilisé avec des encodeurs compatibles Whip tels que OBS pour publier sur IVS en streaming en temps réel. Le WHIP (WebRTC-HTTP Ingestion Protocol) est un projet de l'IETF développé pour normaliser l'ingestion du WebRTC. Consultez la nouvelle page sur OBS et WHIP Support.

1 février 2024

SDK de diffusion Amazon IVS : Android 1.14.1, iOS 1.14.1, Web 1.8.0 (diffusion en temps réel)

Plateforme	Téléchargements et modifications
SDK de diffusion Web 1.8.0	Documentation de référence : https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • Le codage en couches avec diffusion simultanée est désormais désactivé par défaut. • Correction d'un problème en raison duquel une instance de stage ne se déconnectait pas correctement lorsqu'une étape était supprimée ou lorsqu'un participant était déconnecté du serveur. Le SDK émet désormais un `STAGE_CONNECTION_STATE_CHANGED` événement dont l'état est `DISCONNECTED` (au lieu de `ERRORED` et then`CONNECTING`).

Plateforme	Téléchargements et modifications
	• Problème résolu : la publication échouait en cas de mise à jour de la stratégie avec des pistes audio ou vidéo vides.
SDK de diffusion Android 1.14.1	Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast- docs /1.14.1/android • Le codage en couches avec diffusion simultanée est désormais désactivé par défaut. • Mis à jour `libWebRTC` de M108 à M119. • Correction de plusieurs crashs pour améliorer la stabilité globale. • Ajout du support pour la publication stéréo. Cela peut être activé par le biais de l'`StageAudioConfiguration` objet. • Correction d'un bug qui provoquait l'afficha ge d'un fil noir par les participants après avoir rejoint une session. • `libWebRTC` Références internes mises à jour pour éviter les conflits de symboles lorsque d'autres `libWebRTC` versions sont incluses dans la même application hôte.

Plateforme	Téléchargements et modifications
SDK de diffusion iOS 1.14.1	Téléchargez pour le streaming en temps réel : https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip Documentation de référence : https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios • Le codage en couches avec diffusion simultanée est désormais désactivé par défaut. • Mis à jour `libWebRTC` de M108 à M119. • Correction de plusieurs crashs pour améliorer la stabilité globale. • Ajout du support pour la publication stéréo. Cela peut être activé via `IVSLocalStageStreamAudioConfiguration` . • Correction d'un crash lors de l'activation du mode audio uniquement pour les autres participants. • TTV améliorée et taille binaire réduite.

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
arm64-v8a	5,223 MB	13,118 MB
armeabi-v7a	4,524 MB	9,134 MB
x86_64	5,418 MB	13,955 MB
x86	5,61 MB	14,369 MB

Taille du kit SDK de diffusion : iOS

Architecture	Taille compressée	Taille non compressée
arm64	3,350 MB	7,790 MB

3 janvier 2024

SDK de diffusion Amazon IVS : Android 1.13.4, iOS 1.13.4, Web 1.7.0 (diffusion en temps réel)

Plateforme	Téléchargements et modifications
SDK de diffusion Web 1.7.0	Documentation de référence : https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • Amélioré time-to-video pour les abonnés rejoignant les stages. • Suppression de la `minAudioBitrateKbps` propriété (elle n'était pas utilisée). • Restauration réseau améliorée en cas de panne ou de modification d'Internet.
SDK de diffusion Android 1.13.4	Documentation de référence : https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.4/android • StageAudioConfiguration permet désormais de définir si l'annulation de l'écho doit être activée.
SDK de diffusion iOS 1.13.4	Téléchargez pour le streaming en temps réel : https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip

Plateforme	Téléchargements et modifications
	Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast- docs /1.13.4/ios • Sur iOS, nous avons amélioré le moteur audio pour l'enregistrement et la lecture en mettant l'accent sur la stabilité et la récupérabilité. Cela améliore la prise en charge des changements d'itinéraire en cours d'utilisation, améliore le taux de récupération de la batterie dans les cas extrêmes et réduit le nombre de blocages du thread principal. • Correction d'un problème à cause duquel le microphone pouvait rester actif même après avoir été détaché d'une scène, laissant l'indicateur de confidentialité iOS activé. (Le SDK ne traitait pas le son entrant à ce moment-là.)

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
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

Taille du kit SDK de diffusion : iOS

Architecture	Taille compressée	Taille non compressée
arm64	3,45 Mo	7,84 Mo

7 décembre 2023

Nouvelles CloudWatch métriques

Nous avons renommé la métrique PacketLoss (Stage) en DownloadPacketLoss (Stage). Nous avons également publié des CloudWatch statistiques supplémentaires pour le streaming en temps réel IVS :

- DownloadPacketLoss (Étape, participant)
- DroppedFrames (Étape, participant)
- SubscribeBitrate (Scène, participant,MediaType)

Pour plus d'informations, consultez Surveillance du streaming en temps réel IVS.

4 décembre 2023

SDK de diffusion Amazon IVS : Android 1.13.2 et iOS 1.13.2 (diffusion en temps réel)

Plateforme	Téléchargements et modifications
Tous les mobiles (Android et iOS)	• La configuration de suppression du bruit est disponible pour être activée/désactivée par les développeurs pour la publication.
SDK de diffusion Android 1.13.2	Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/android

Plateforme	Téléchargements et modifications
	• Amélioration du temps de chargement de la vidéo (TTV) lorsque vous rejoignez la première étape d'une session.
SDK de diffusion iOS 1.13.2	Téléchargez pour le streaming en temps réel : https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip Documentation de référence : https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/ios • Aucune modification n'a été apportée au SDK en temps réel.

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
arm64-v8a	5,177 Mo	13,01 Mo
armeabi-v7a	4,485 Mo	9,045 Mo
x86_64	5,352 Mo	13,808 Mo
x86	5,547 Mo	14,192 Mo

Taille du kit SDK de diffusion : iOS

Architecture	Taille compressée	Taille non compressée
arm64	3,45 Mo	7,82 Mo

21 novembre 2023

SDK de diffusion Amazon IVS : Android 1.13.1 (Streaming en temps réel)

Plateforme	Téléchargements et modifications
SDK de diffusion Android 1.13.1	Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast- docs /1.13.1/android • Correction d'un problème qui provoquait un arrêt brusque lorsque l'on quittait, publiait et rejoignait rapidement la même scène.

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
arm64-v8a	5,177 Mo	13,102 Mo
armeabi-v7a	4,485 Mo	9,046 Mo
x86_64	5,353 Mo	13,809 Mo
x86	5,547 Mo	14,192 Mo

17 novembre 2023

SDK de diffusion Amazon IVS : Android 1.13.0 et iOS 1.13.0 (diffusion en temps réel)

Plateforme	Téléchargements et modifications
Tous les mobiles (Android et iOS)	• Optimisations du streaming mises à jour. Entre autres, la fonctionnalité « Streaming

Plateforme	Téléchargements et modifications
	adaptatif : encodage en couches avec diffusion simultanée » nécessite désormais une inscription explicite et n'est prise en charge que dans les versions récentes du SDK. • Amélioration de la stabilité des scènes en réduisant l'occurrence des pannes rares. • Amélioration du temps de chargement de la vidéo (TTV) lorsque vous rejoignez une scène. • Amélioration de l'expérience avec les appareils Bluetooth. • Optimisation de l'utilisation du processeur et de la mémoire du SDK et réduction de la taille de la bibliothèque. • Ajout de la classe `StageAudioManager` , qui peut être utilisée pour définir les paramètres de capture et de lecture audio, y compris les préréglages pour la communica tion vocale, la lecture multimédia, etc. Pour plus de détails, consultez la nouvelle page, SDK de diffusion IVS : Modes audio mobiles. • Ajout d'une nouvelle fonction `requestQu alityStats` qui permet d'afficher des événements de qualité structurés à partir des statistiques WebRTC. • Ajout d'une nouvelle fonctionnalité pour mettre à jour le débit audio. Elle est définie sur des objets `LocalStageStream` , tout comme la configuration vidéo, mais via un nouvel objet de configuration audio.

Plateforme	Téléchargements et modifications

SDK de diffusion Android 1.13.0	Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast- docs /1.13.0/android
	• Toutes les méthodes de l'interface `StageRenderer` sont désormais facultati ves.
	• Ajout de la prise en charge de la prévisual isation basée sur `Surfaceview` pour de meilleures performances. Les méthodes `getPreview` existantes dans `Session` et `StageStream` continuent de renvoyer une sous-classe de `TextureView` , mais cela pourrait changer dans une future version du SDK.
	  • Si votre application dépend de `TextureVi ew` spécifiquement, vous pouvez continuer sans apporter de modificat ion. Vous pouvez également passer de `getPreview` à `getPreviewTextureV iew` pour vous préparer à la modification éventuelle de ce que renvoie la valeur par défaut `getPreview` .
	  • Si votre application n'en a pas besoin de `TextureView` spécifiquement, nous vous recommandons de passer à l'option `getPreviewSurfaceView` pour réduire l'utilisation du processeur et de la mémoire.
	• Le SDK implémente désormais un nouveau type d'aperçu appelé `ImagePrev iewSurfaceTarget` . Ce dernier fonctionne avec l'objet Surface Android fourni par l'application. Il ne s'agit pas d'une

Plateforme	Téléchargements et modifications
	sous-classe d'Android View, qui offre une meilleure flexibilité. • Correction du cas où le rappel `onFrame` d'un participant distant était appelé au mauvais moment avec la mauvaise taille. • `SurfaceSource # getInputSurface` est désormais annoté avec `@Nullable` . Votre code doit le vérifier avant de l'utiliser. • Ajout de `UserId` et `attributes` à `ParticipantInfo` . Les propriétés `UserId` et `attributes` sont intégrées au jeton et les applications peuvent les récupérer avec `ParticipantInfo` à chaque fois qu'un participant se joint. • La capture de caméra et le rendu d'aperçu sont désormais définis par défaut sur 720 x 1280 ou la résolution de diffusion (selon la valeur la plus élevée) à 15 images par seconde. Pour régler la résolution et/ou les images par seconde, utilisez `StageVideoConfiguration # setCameraCaptureQuality` . • Si `IllegalArgumentException` est lancée lors de la définition des propriétés de configuration, elle inclura désormais la valeur fournie dans le message d'exception.

Plateforme	Téléchargements et modifications
SDK de diffusion iOS 1.13.0	Téléchargez pour le streaming en temps réel : https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentation de référence : https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.0/ios • Le problème selon lequel le SDK ne modifiait pas la configuration vidéo si celle-ci était mise à jour avant la diffusion a été résolu. • Intégration du correctif Google pour une vulnérabilité de sécurité dans LibVPX (CVE-2023-5217). (Notez que le SDK Android n'avait besoin d'aucune modification pour ce problème.) • Les applications qui utilisent d'autres bibliothèques incluant `libWebRTC` n'auront plus de conflits avec le SDK de diffusion IVS. • Toutes les méthodes du protocole `IVSStageRenderer` sont désormais marquées `@optional` . • Les microphones et caméras renvoyés par nos SDK sont maintenant soumis à un ordre de tri garanti, comme indiqué dans les SDK eux-mêmes. • Plusieurs caméras peuvent désormais avoir une valeur égale à `true` pour leur propriété `isDefault` , une pour chaque position déterminée par le système d'exploitation. • Ajout de `IVSStageAudioManager` , ce qui permet un contrôle précis du `AVAudioSession` sous-jacent afin de permettre une

Plateforme	Téléchargements et modifications
	plus grande variété de cas d'utilisation de la fonctionnalité Scènes. • Ajout de `UserId` à `ParticipantInfo`

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
arm64-v8a	5,17 Mo	13,00 Mo
armeabi-v7a	4,48 Mo	9,04 Mo
x86_64	5,35 Mo	13,80 Mo
x86	5,54 Mo	14,18 Mo

Taille du kit SDK de diffusion : iOS

Architecture	Taille compressée	Taille non compressée
arm64	3,45 Mo	7,84 Mo

16 novembre 2023

Enregistrement composite

Cette nouvelle fonctionnalité permet d'enregistrer la vue composite d'une scène IVS dans un compartiment Amazon S3. Pour plus d'informations, consultez :

- Enregistrement composite : ceci est une nouvelle page.
- Premiers pas avec le streaming en temps réel IVS : nous avons ajouté des points de terminaison S3 à la politique dans « Configurer les autorisations IAM ».

- Service Quotas : nous avons ajouté des quotas de taux d'appel pour les nouveaux points de terminaison.
- Référence de l'API IVS Real-Time Streaming — Nous avons ajouté 4 StorageConfiguration points de terminaison et 7 objets (DestinationDetail,, S3 RecordingConfigurationDestinationConfiguration, S3Detail, S3StorageConfiguration,,). StorageConfiguration StorageConfigurationSummary Nous avons également modifié 3 objets (Composition, Destination, DestinationConfiguration) ; cela affecte la GetComposition réponse et la StartComposition demande et la réponse.

16 novembre 2023

Montage côté serveur

Le montage côté serveur IVS permet aux clients de décharger le montage et la diffusion d'une scène IVS vers un service géré par IVS. Le montage côté serveur et la diffusion RTMP vers un canal sont invoquées via les points de terminaison du plan de contrôle IVS situés dans la région d'origine de la scène. Pour plus d'informations, consultez :

- Premiers pas avec le streaming en temps réel IVS : nous avons ajouté des points de terminaison SSC à la politique dans « Configurer les autorisations IAM ».
- Utilisation d'Amazon EventBridge avec IVS Real-Time Streaming — Nous avons ajouté de nouvelles statistiques.
- Montage côté serveur : ce nouveau document inclut un aperçu et des instructions de configuration.
- Service Quotas (diffusion en temps réel) : nous avons ajouté de nouvelles limites de débit d'appels et d'autres quotas.
- Référence de l'API de streaming en temps réel — Nous avons ajouté 8 EncoderConfiguration points de composition et 11 objets (Composition ChannelDestinationConfiguration CompositionSummary, Destination DestinationConfiguration, DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration, LayoutConfiguration, et vidéo).

Dans le Guide de l'utilisateur du streaming à faible latence Amazon IVS, consultez :

- Activation de plusieurs hôtes sur un flux IVS : nous avons ajouté « Diffuser une scène : montage côté serveur contre côté client » et mis à jour "4. Diffuser la scène.

16 octobre 2023

SDK de diffusion Amazon IVS : Web 1.6.0 (Streaming en temps réel)

Plateforme	Téléchargements et modifications
SDK de diffusion Web 1.6.0	Documentation de référence : https://aws.github.io/ amazon-ivs-web-broadcast /docs/ sdk-reference • Time-to-Video (TTV) amélioré. • Ajout de la configuration `maxAudioBitrate` , prenant en charge jusqu'à 128 kbps de canaux audio mono ou stéréo.

12 octobre 2023

Nouveaux CloudWatch indicateurs et données sur les participants

Nous avons publié CloudWatch des statistiques pour le streaming en temps réel IVS. Pour plus d'informations, consultez Surveillance du streaming en temps réel IVS.

Nous avons également ajouté six champs à l'API d'objet participant : `browserName`, `browserVersion`, `ispName`, `osName`, `osVersion` et `sdkVersion`. Cela influe sur la GetParticipant réponse. Consultez la Référence de l'API de streaming en temps réel Amazon IVS.

12 octobre 2023

SDK de diffusion Amazon IVS : Android 1.12.1 (Streaming en temps réel)

Plateforme	Téléchargements et modifications
Kit SDK de diffusion Android 1.12.1	Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast- docs /1.12.1/android

Plateforme	Téléchargements et modifications
	• Correction d'un bug lors duquel l'appel de `BroadcastSession.setListener` provoquait une erreur.

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
arm64-v8a	5,853 Mo	16,375 Mo
armeabi-v7a	4,895 Mo	10,803 Mo
x86_64	6,149 Mo	17,318 Mo
x86	6,328 Mo	17,186 Mo

14 septembre 2023

SDK de diffusion Amazon IVS : Web 1.5.2 (Streaming en temps réel)

Plateforme	Téléchargements et modifications
SDK de diffusion Web 1.5.2	Documentation de référence : https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • Correction d'un bogue qui empêchait la republication avec `refreshStrategy` lorsque l'état publié entrait dans un état `ERRORED`.

23 août 2023

Kit SDK de diffusion Amazon IVS : Web 1.5.1, Android 1.12.0 et iOS 1.12.0 (diffusion en temps réel)

Plateforme	Téléchargements et modifications
SDK de diffusion Web 1.5.1	Documentation de référence : https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • Correction d'un bug avec les types Maybe internes sur TypeScript 5. • Ajout d'une meilleure détection pour le support de la diffusion simultanée. • Correction de deux conditions de concurrence avec `refreshStrategy` lors de la publication. • Correction d'une condition de concurrence avec `refreshStrategy` lors de la mise à jour des participants auxquels s'abonner.
Tous les mobiles (Android et iOS)	• Correction d'un problème rare où l'action de diffusion n'est jamais terminée. • Amélioration de la stabilité des scènes en réduisant l'occurrence des pannes rares. • Amélioration de la stabilité des étapes en résolvant les problèmes de conditions de concurrence causés par des joints/quittés rapides. • Ajout d'une nouvelle méthode `setOnFrameCallback` sur `ImageDevice` . Elle permet d'observer les images qui passent à travers l'appareil lui-même, ce qui fournit des informations sur le rapport hauteur/l

Plateforme	Téléchargements et modifications
	argeur des dernières images. Cette méthode peut également être utilisée pour détecter le moment où la première image est rendue pour un participant distant dans une étape.
Kit SDK de diffusion Android 1.12.0	Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast- docs /1.12.0/android • Android 9 est désormais pris en charge. • Utilisation et performances du processeur améliorées.
SDK de diffusion iOS 1.12.0	Téléchargez pour le streaming en temps réel : https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentation de référence : https:// aws.github.io/ amazon-ivs-broadcast- docs /1.12.0/ios • Correction de la signature de `IVSDeviceDiscovery.createAudioSourceWithName` pour renvoyer une `IVSCustomAudioSource` au lieu de `IVSCustomImageSource` .

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
arm64-v8a	5,853 Mo	16,375 Mo
armeabi-v7a	4,895 Mo	10,803 Mo
x86_64	6,149 Mo	17,318 Mo

Architecture	Taille compressée	Taille non compressée
x86	6,328 Mo	17,186 Mo

Taille du kit SDK de diffusion : iOS

Architecture	Taille compressée	Taille non compressée
arm64	5,06 Mo	10,92 Mo

7 août 2023

SDK de diffusion Amazon IVS : Web 1.5.0, Android 1.11.0 et iOS 1.11.0

Plateforme	Téléchargements et modifications
SDK de diffusion Web 1.5.0	Documentation de référence : https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • Ajout de Simulcast : lorsqu'elle est activée, cette fonctionnalité permet au diffuseur de publication d'envoyer des couches vidéo de haute et de faible qualité. Les abonnés sélectionnent automatiquement la qualité optimale en fonction des conditions de leur réseau. Consultez la section Optimisation de contenu multimédia.
Tous les mobiles (Android et iOS)	Ajout de Simulcast : lorsqu'elle est activée, cette fonctionnalité permet au diffuseur de publication d'envoyer des couches vidéo de haute et de faible qualité. Les abonnés sélectionnent automatiquement la qualité optimale en fonction des conditions de

Plateforme	Téléchargements et modifications
	leur réseau. Consultez la section « Activer/désactiver le codage en couches avec Simulcast » dans les guides du SDK de diffusion Android et iOS.
SDK de diffusion Android 1.11.0	Documentation de référence : https://aws.github.io/ amazon-ivs-broadcast-docs /1.11.0/android • Correction d'un problème où la création de nombreuses scènes entraînait un plantage. (Le nombre exact de scènes dépend de l'appareil.)
SDK de diffusion iOS 1.11.0	Téléchargement pour le streaming en temps réel : https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentation de référence : https://aws.github.io/ amazon-ivs-broadcast-docs /1.11.0/ios • Correction de la signature de `IVSDeviceDiscovery.createAudioSourceWithName` pour renvoyer `IVSCustomAudioSource` au lieu de `IVSCustomImageSource` .

Taille du kit SDK de diffusion : Android

Architecture	Taille compressée	Taille non compressée
arm64-v8a	5,811 Mo	16,186 Mo
armeabi-v7a	4,857 Mo	10,646 Mo

Architecture	Taille compressée	Taille non compressée
x86_64	6,108 Mo	17,122 Mo
x86	6,289 Mo	16,994 Mo

Taille du kit SDK de diffusion : iOS

Architecture	Taille compressée	Taille non compressée
arm64	5,030 Mo	10,810 Mo

7 août 2023

Streaming en temps réel

Le streaming en temps réel Amazon Interactive Video Service (IVS) vous permet de diffuser des diffusions en direct avec une latence qui peut être inférieure à 300 millisecondes entre l'hôte et le spectateur.

Des modifications majeures de la documentation accompagnent cette version. La page d'accueil de la documentation IVS comporte désormais des sections distinctes pour le streaming en temps réel et le streaming à faible latence. Chaque section possède son propre guide de l'utilisateur et sa propre référence d'API. Pour plus de détails sur la documentation, consultez la section Historique du document (pour les deux modifications de la documentation respectives : en temps réel et à faible latence). Pour le streaming en temps réel, commencez par le Guide de l'utilisateur du streaming en temps réel IVS et la Référence de l'API de streaming en temps réel IVS.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.