Guida per l'utente dello streaming in tempo reale

Amazon IVS

Amazon IVS: Guida per l'utente dello streaming in tempo reale

Table of Contents

Cos'è lo streaming in tempo reale di Amazon IVS?

Lo streaming in tempo reale di Amazon Interactive Video Service (IVS) offre tutto ciò di cui hai bisogno per aggiungere audio e video in tempo reale alle tue applicazioni.

Efficacia:

- Latenza in tempo reale: crea applicazioni per casi d'uso sensibili alla latenza, aiutando i tuoi spettatori a rimanere connessi e coinvolti con lo streaming in tempo reale di IVS. Offri streaming live con una latenza che può essere inferiore a 300 millisecondi dall'host allo spettatore.
- Elevata concorrenza: sblocca il potenziale delle interazioni su larga scala con lo streaming IVS in tempo reale. Soddisfa un pubblico fino a 10.000 spettatori e consenti a un massimo di 12 presentatori (host) di salire sul palco virtuale.
- Ottimizzato per dispositivi mobili: lo streaming in tempo reale di IVS è ottimizzato per i casi d'uso mobili, soddisfacendo una vasta gamma di dispositivi e funzionalità di rete. Integrando gli SDK di trasmissione Amazon IVS per Android e iOS, i tuoi utenti possono interagire come host o spettatori, godendo di streaming live di alta qualità sui propri dispositivi mobili.

Casi d'uso:

- Spot per gli ospiti: crea applicazioni che consentano agli host di promuovere gli ospiti "sul palco", trasformando gli spettatori in host per interazioni in tempo reale.
- Modalità Versus (VS): produci esperienze con competizioni parallele e consenti agli spettatori di guardare gli host competere in tempo reale.
- Sale audio: invita gli ascoltatori a partecipare alla conversazione come ospiti e promuovi un coinvolgimento più profondo nelle tue sale audio.
- Aste video in diretta: trasforma le aste in eventi video interattivi e mantienine l'entusiasmo e l'integrità con una latenza in tempo reale.

Oltre alla documentazione del prodotto qui, consulta il sito dedicato https://ivs.rocks/ per sfogliare i contenuti pubblicati (demo, esempi di codice, post di blog), stimare i costi e sperimentare Amazon IVS attraverso demo live.

Soluzione globale, controllo regionale

Streaming e visualizzazione sono globali

Puoi utilizzare Amazon IVS per trasmettere in streaming agli spettatori di tutto il mondo:

- Quando esegui lo streaming, Amazon IVS inserisce automaticamente i video in una posizione vicina a te.
- Gli spettatori possono guardare i tuoi streaming live a livello globale.

Questo è un altro modo per dire che il "piano dati" è globale. Il piano dati si riferisce a streaming/acquisizione e visualizzazione.

Il controllo è regionale

Mentre il piano dati Amazon IVS è globale, il "piano di controllo" è regionale. Il piano di controllo si riferisce alla console, all'API e alle risorse (fasi) di Amazon IVS.

Un altro modo per dirlo è che Amazon IVS è un "servizio AWS regionale". In altre parole, le risorse Amazon IVS in ogni regione sono indipendenti da risorse simili in altre regioni. Ad esempio, una fase creata in una regione è indipendente dalle fasi create in altre regioni.

Quando si utilizzano risorse (ad esempio, si crea una fase), è necessario specificare la regione in cui verranno create. Successivamente, quando si gestiscono le risorse, sarà necessario farlo dalla stessa regione in cui sono state create.

Se utilizzi...	Si specifica la regione per...
Console Amazon IVS	Tramite l'elenco a discesa Seleziona una regione nella parte superiore destra della barra di navigazione.
API Amazon IVS	Utilizzo dell'endpoint di servizio appropriato. Consulta la Documentazione di riferimento delle API di streaming in tempo reale Amazon IVS. Se accedi all'API tramite un SDK, configura il parametro `region` dell'SDK. Consulta Strumenti per creare su AWS.
CLI AWS	Una delle seguenti opzioni:

Se utilizzi...	Si specifica la regione per...
	• Aggiungendo `--region <aws-region>` al comando della CLI. • Inserendo la regione nel file di configurazione AWS locale.

Ricorda che, indipendentemente dalla regione in cui è stata creata una fase, puoi trasmettere in streaming su Amazon IVS da qualsiasi luogo e gli spettatori possono guardare da qualsiasi luogo.

Guida introduttiva allo streaming in tempo reale di IVS

Questo documento illustra i passaggi necessari per integrare lo streaming in tempo reale di Amazon IVS nella tua app.

Argomenti

Introduzione

Prerequisiti

Prima di utilizzare lo streaming in tempo reale per la prima volta, completa le seguenti attività. Per le istruzioni, consulta Guida introduttiva allo streaming a bassa latenza di IVS.

- Creazione di un account AWS
- Configurazione degli utenti root e amministrativi

Altri riferimenti

- Riferimento all'SDK di trasmissione Web IVS
- Riferimento all'SDK di trasmissione IVS per Android
- Riferimento all'SDK di trasmissione IVS per iOS
- Riferimento all'API di streaming in tempo reale IVS

Terminologia dello streaming in tempo reale

Termine	Descrizione
Stage	Uno spazio virtuale in cui i partecipanti possono scambiarsi audio e video in tempo reale.
Host	Un partecipante che invia un video locale alla fase.
Visualizzatore	Un partecipante che riceve un video degli host.
Partecipante	Un utente connesso alla fase come host (presentatore) o spettatore.
Token dei partecipanti	Un token che autentica un partecipante quando entra in una fase.
SDK di trasmissione	Una libreria client che consente ai partecipanti di inviare e ricevere video.

Panoramica delle fasi

1. the section called “Impostazione delle autorizzazioni IAM”: crea una policy di AWS Identity and Access Management (IAM) che fornisce agli utenti un set di autorizzazioni di base e assegna tale policy agli utenti.
2. Crea una fase: crea uno spazio virtuale in cui i partecipanti possono scambiarsi video in tempo reale.
3. Distribuisci i token ai partecipanti: invia token ai partecipanti in modo che possano unirsi alla tua fase.

4. Integra l'SDK di trasmissione IVS: aggiungi l'SDK di trasmissione alla tua app per consentire ai partecipanti di inviare e ricevere video: the section called “App”, the section called “Android” e the section called “iOS”.
5. Pubblica e sottoscrivi video: invia il tuo video alla fase e ricevi video da altri host: the section called “App”, the section called “Android” e the section called “iOS”.

Impostazione delle autorizzazioni IAM

Successivamente, sarà necessario creare una policy AWS Identity and Access Management (IAM) che fornisca agli utenti un set di autorizzazioni di base (ad esempio per creare una fase di Amazon IVS e i token dei partecipanti) e assegnare tale policy agli utenti. È possibile assegnare le autorizzazioni durante la creazione di un nuovo utente o aggiungere le autorizzazioni a un utente esistente. Entrambe le procedure sono riportate di seguito.

Per ulteriori informazioni (ad esempio, per informazioni sugli utenti e sulle policy IAM, su come collegare una policy a un utente e su come vincolare ciò che gli utenti possono fare con Amazon IVS), consultare:

- Creazione di un utente IAM nella Guida per l'utente di IAM
- Le informazioni contenute in Sicurezza di Amazon IVS su IAM e "Policy gestite per IVS".
- Le informazioni IAM in Sicurezza di Amazon IVS

Puoi utilizzare una policy gestita da AWS esistente per Amazon IVS o crearne una nuova che personalizzi le autorizzazioni che desideri concedere a un insieme di utenti, gruppi o ruoli. Entrambi gli approcci sono descritti di seguito.

Utilizza una policy esistente per le autorizzazioni IVS

Nella maggior parte dei casi, ti consigliamo di utilizzare una policy gestita da AWS per Amazon IVS. Sono descritte in dettaglio nella sezione Policy gestite per IVS di Sicurezza di IVS.

- Utilizza la policy gestita da AWS `IVSReadOnlyAccess` per consentire agli sviluppatori di applicazioni di accedere a tutti gli endpoint API IVS Get and List (sia per lo streaming a bassa latenza che in tempo reale).
- Utilizza la policy gestita da AWS `IVSFullAccess` per consentire agli sviluppatori di applicazioni di accedere a tutti gli endpoint API IVS (sia per lo streaming a bassa latenza che in tempo reale).

Facoltativo: creazione di una policy personalizzata per le autorizzazioni Amazon IVS

Completa la procedura riportata di seguito.

1. Accedere alla Gestione della Console AWS e aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel riquadro di navigazione, selezionare Policies (Policy) e Create Policy (Crea policy). Si apre una finestra Specifica le autorizzazioni.
3. Nella finestra Specifica autorizzazioni seleziona la scheda JSON, quindi copia e incolla la seguente policy IVS nell'area ti testo Editor delle policy. (La policy non include tutte le azioni di Amazon IVS. È possibile aggiungere/eliminare (Consentire/negare) le autorizzazioni di accesso agli endpoint in base alle esigenze. Consulta Riferimento all'API di streaming in tempo reale 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. Sempre nella finestra Specifica autorizzazioni, scegli Successivo (scorri fino alla fine della finestra per visualizzare questa opzione). Si apre una finestra Rivedi e crea.
5. Nella finestra Rivedi e crea inserisci un Nome per la policy e, facoltativamente, aggiungi una Descrizione. Prendi nota del nome della policy poiché sarà necessario per creare gli utenti (di seguito). Scegli Create policy (Crea policy) nella parte inferiore della finestra.
6. Tornerai alla finestra della console IAM, dove dovresti vedere un banner che conferma che la tua nuova policy è stata creata.

Creare un Nuovo utente e Aggiungere autorizzazioni

Chiavi di accesso utente IAM

Le chiavi di accesso IAM sono composte da un ID chiave di accesso e una chiave di accesso segreta. Vengono utilizzati per firmare le richieste a livello di programmazione che fai ad AWS. In mancanza di chiavi di accesso, puoi crearle utilizzando la Console di gestione AWS. Come best practice, non utilizzare le chiavi di accesso dell'utente root.

L'unico momento in cui è possibile visualizzare o scaricare la chiave di accesso segreta è durante la creazione delle chiavi di accesso. Non sarà possibile recuperarle successivamente. Tuttavia, è possibile creare nuove chiavi di accesso in qualsiasi momento; è necessario disporre delle autorizzazioni per effettuare le operazioni IAM richieste.

Conserva sempre le chiavi di accesso in modo sicuro. Non condividerle mai con terze parti (anche se sembra che una richiesta provenga da Amazon). Per ulteriori informazioni, consulta Gestione delle chiavi di accesso per gli utenti IAM nella Guida per l'utente di IAM .

Procedura

Completare la procedura riportata di seguito.

1. Nel riquadro di navigazione, seleziona Utenti, quindi Crea utente. Si apre una finestra Specifica i dettagli dell'utente.
2. Nella finestra Specifica i dettagli dell'utente:
   a. Sotto a Dettagli dell'utente, digita il nuovo Nome utente da creare.
   b. Seleziona la casella di controllo Console di gestione AWS.
   c. Per Console password (Password della console), seleziona Autogenerated password (Password generata automaticamente).
   d. Seleziona la casella di controllo Gli utenti devono creare una nuova password all'accesso successivo.
   e. Seleziona Avanti. Si apre una finestra Imposta autorizzazioni.
3. Sotto a Imposta autorizzazioni, seleziona Collega direttamente le policy esistenti. Si apre una finestra Policy di autorizzazione.
4. Nella casella di ricerca, inserisci il nome di una policy IVS (una policy gestita da AWS o personalizzata creata in precedenza). Una volta trovato, seleziona la casella per selezionare la policy.
5. Seleziona Successivo (nella parte inferiore della finestra). Si apre una finestra Rivedi e crea.
6. Nella finestra Rivedi e crea, conferma che tutti i dettagli dell'utente siano corretti, quindi scegli Crea utente (nella parte inferiore della finestra).
7. Si apre la finestra Recupera password, contenente i Dettagli di accesso alla console. Salva queste informazioni in modo sicuro per l'utilizzo futuro. Al termine, seleziona Torna all'elenco utenti.

Aggiungere autorizzazioni a un utente esistente

Completa la procedura riportata di seguito.

1. Accedere alla Gestione della Console AWS e aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel pannello di navigazione, scegliere Utenti, quindi un nome utente esistente da aggiornare. (Scegli il nome facendo clic su di esso; non selezionare la casella di selezione.)
3. Nella pagina Riepilogo, nella scheda Autorizzazioni, seleziona Aggiungi autorizzazioni. Si apre una finestra Aggiungi autorizzazioni.

4. Seleziona Attach existing policies directly (Collega direttamente le policy esistenti). Si apre una finestra Policy di autorizzazione.
5. Nella casella di ricerca, inserisci il nome di una policy IVS (una policy gestita da AWS o personalizzata creata in precedenza). Una volta trovata la policy, seleziona la casella per selezionarla.
6. Seleziona Successivo (nella parte inferiore della finestra). Si apre una finestra Rivedi.
7. Nella finestra Rivedi, seleziona Aggiungi autorizzazioni (nella parte inferiore della finestra).
8. Nella pagina di riepilogo, verifica che la policy IVS sia stata aggiunta.

Creazione di uno stage

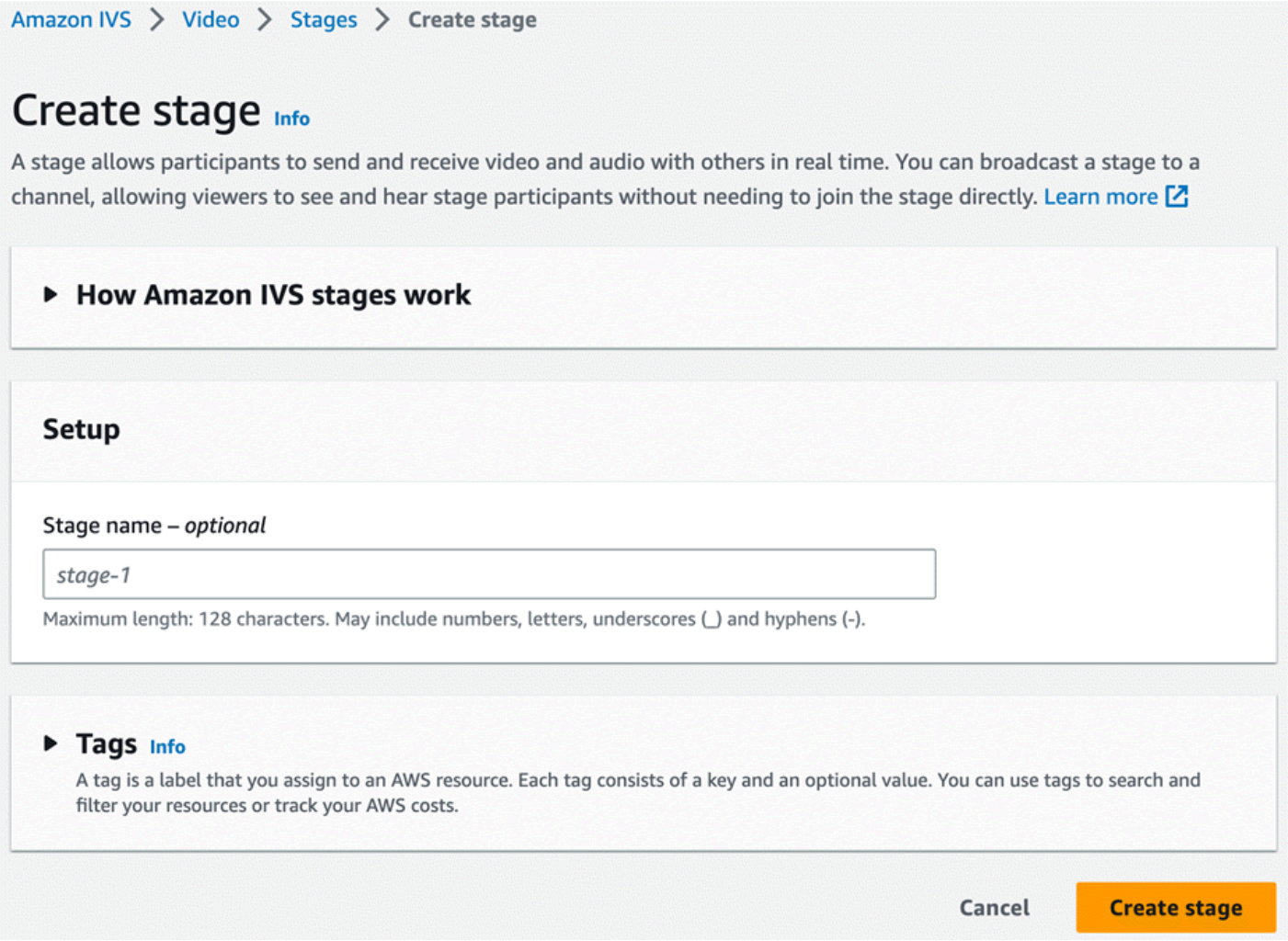
Una fase è uno spazio virtuale in cui i partecipanti possono scambiarsi video in tempo reale. È la risorsa fondamentale dell'API dello streaming in tempo reale. È possibile creare uno stage utilizzando la console o l' CreateStage endpoint.

Si consiglia, laddove possibile, di creare una nuova fase per ogni sessione logica e di eliminarla una volta completate le operazioni piuttosto che mantenere le vecchie fasi per un eventuale riutilizzo. Se le risorse obsolete (vecchie fasi, da non riutilizzare) non vengono cancellate, è probabile che tu raggiunga più velocemente il limite del numero massimo di fasi.

Istruzioni per la console

1. Aprire la Console Amazon IVS.

   L'accesso alla console Amazon IVS è possibile anche dalla Console di gestione AWS.
2. Nel riquadro di navigazione a sinistra, seleziona Fase, quindi seleziona Crea fase. Viene visualizzata la finestra Crea fase.



3. Facoltativamente, inserisci un Nome fase. Seleziona Crea fase per creare la fase. Viene visualizzata la pagina dei dettagli della fase relativa al nuova fase.

Istruzioni per la CLI

Per installare AWS CLI, consulta Installazione o aggiornamento della versione più recente di AWS CLI.

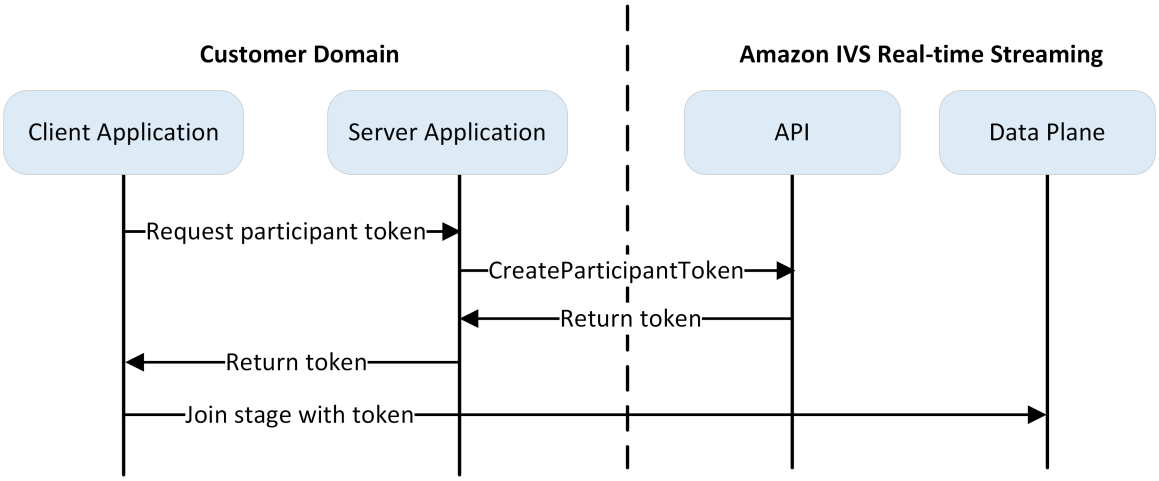
Ora puoi usare la CLI per creare e gestire le risorse. L'API della fase si trova nello spazio dei nomi ivs-realtime. Ad esempio, per creare uno stage:

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

La risposta è:

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

Distribuzione dei token di partecipazione



Ora che hai una fase, devi creare e distribuire i token ai partecipanti per consentire loro di partecipare alla fase e iniziare a inviare e ricevere video.

Come mostrato sopra, un'applicazione client richiede all'applicazione server un token e l'applicazione server chiama CreateParticipantToken utilizzando una richiesta firmata AWS SDK o SigV4. Poiché le credenziali AWS vengono utilizzate per chiamare l'API, il token deve essere generato in un'applicazione sicura lato server, non nell'applicazione lato client.

Quando crei un token di un partecipante, puoi facoltativamente specificare le funzionalità abilitate da quel token. L'opzione predefinita è `PUBLISH` e `SUBSCRIBE`, che consente al partecipante di inviare e ricevere audio e video, ma è possibile emettere token con un sottoinsieme di funzionalità. Ad esempio, puoi emettere un token con la sola capacità `SUBSCRIBE` per i moderatori. In tal caso, i moderatori potrebbero vedere i partecipanti che inviano video ma non inviare a loro volta un video.

Puoi creare token per i partecipanti tramite la console o l'interfaccia a riga di comando per test e sviluppo, ma molto probabilmente vorrai crearli con l'SDK AWS nel tuo ambiente di produzione.

Avrai bisogno di un modo per distribuire i token dal server a ciascun client (ad esempio, tramite una richiesta API). Questa funzionalità non è disponibile. Per questa guida, puoi semplicemente copiare e incollare i token nel codice cliente come riportato nei seguenti passaggi.

Importante: tratta i token come opachi; ad esempio, non crea funzionalità basate sul contenuto dei token. Il formato dei token potrebbe cambiare in futuro.

Istruzioni per la console

1. Passa alla fase che hai creato nel passaggio precedente.
2. Seleziona Crea un token per i partecipanti. Viene visualizzata la finestra Crea un token del partecipante.
3. Inserisci un ID utente da associare al token. Può essere qualsiasi testo codificato in UTF-8.
4. Seleziona Crea un token per i partecipanti.
5. Copiare il token. Importante: assicurati di salvare il token; IVS non lo memorizza e non potrai recuperarlo in seguito.

Istruzioni per la CLI

La creazione di un token con AWS CLI richiede prima il download e la configurazione della CLI sul computer. Per maggiori dettagli, consultare la Guida per l'utente dell'interfaccia a riga di comando di AWS. Nota: la generazione di token con AWS CLI è utile per scopi di test, ma per l'uso in produzione si consiglia di generare token sul lato server con l'SDK AWS (vedere le istruzioni sopra).

1. Esegui il comando `create-participant-token` con l'ARN della fase. Includi una o tutte le funzionalità seguenti: `"PUBLISH"`, `"SUBSCRIBE"`.

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

2. Questa operazione restituisce un token del partecipante:

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

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

3. Salvare questo token. Ti servirà per unirti alla fase e inviare e ricevere video.

Istruzioni per gli SDK AWS

Per creare i token, è possibile utilizzare l'SDK AWS. Di seguito sono riportate le istruzioni per l'utilizzo dell'SDK. AWS JavaScript

Importante: questo codice deve essere eseguito sul lato server e il relativo output passato al client.

Prerequisito: per utilizzare l'esempio di codice riportato di seguito, è necessario installare il pacchetto client-ivs-realtime aws-sdk/. Per i dettagli, consulta la sezione Guida introduttiva all'SDK AWS per 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);
```

Integrazione dell'SDK di trasmissione IVS

IVS fornisce un SDK di trasmissione per Web, Android e iOS che può essere integrato nella tua applicazione. L'SDK di trasmissione viene utilizzato sia per l'invio che per la ricezione di video. In questa sezione, viene descritto come scrivere una semplice applicazione che consente a due o più partecipanti di interagire in tempo reale. I passaggi seguenti ti guidano nella creazione di un'app chiamata BasicRealTime. Il codice completo dell'app è attivo CodePen e GitHub:

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

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

App

Configurazione dei file

Per iniziare, configura i file creando una cartella e un file HTML e JS iniziale:

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

Puoi installare l'SDK di trasmissione usando un tag di script o npm. Nel nostro esempio verrà utilizzato il tag script per semplicità, ma è facile da modificare se si decide di utilizzare npm in un secondo momento.

Utilizzo di un tag di script

Il Web broadcast SDK è distribuito come JavaScript libreria e può essere recuperato all'indirizzo https://web-broadcast.live-video.net/1.8.0/ amazon-ivs-web-broadcast .js.

Quando viene caricata tramite il tag `<script>`, la libreria espone una variabile globale nell'ambito della finestra denominato `IVSBroadcastClient`.

Utilizzo di npm

Per installare il pacchetto npm:

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

È ora possibile accedere all'oggetto IVS: BroadcastClient

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

Android

Creazione del progetto Android

1. In Android Studio, crea un nuovo progetto.

2. Scegli Attività di viste vuote.

   Nota: in alcune versioni precedenti di Android Studio, l'attività basata sulle viste è detta attività vuota. Se in Android Studio viene visualizzata la finestra Attività vuota ma non mostra l'attività Viste vuote, seleziona Attività vuota. Altrimenti, non selezionare Attività vuota, poiché utilizzeremo le API View (non Jetpack Compose).
3. Assegna un nome al tuo progetto, quindi seleziona Fine.

Installazione dell'SDK di trasmissione

Per aggiungere la libreria di trasmissione di Amazon IVS per Android al proprio ambiente di sviluppo Android, aggiungi la libreria al file `build.gradle` del modulo come mostrato di seguito (per la versione più recente dell'SDK di trasmissione Amazon IVS): Nei progetti più recenti il repository `mavenCentral` potrebbe essere già incluso nel file`settings.gradle`, in tal caso puoi omettere il blocco `repositories`. Per il nostro esempio, dovremo anche abilitare l'associazione dei dati nel blocco `android`.

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
}

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

In alternativa, per installare manualmente l'SDK, scaricare la versione più recente da questo percorso:

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

iOS

Creazione del progetto iOS

1. Crea un nuovo progetto Xcode.
2. Per Piattaforma, seleziona iOS.

3. Per Applicazione, seleziona App.
4. Inserisci il nome del prodotto della tua app, quindi seleziona Successivo.
5. Scegli o passa a una directory in cui salvare il progetto, quindi seleziona Crea.

Successivamente, dovrai inserire l'SDK. Ti consigliamo di integrare l'SDK di trasmissione tramite. CocoaPods In alternativa, esiste la possibilità di aggiungere manualmente il framework al proprio progetto. Entrambi i metodi sono descritti di seguito.

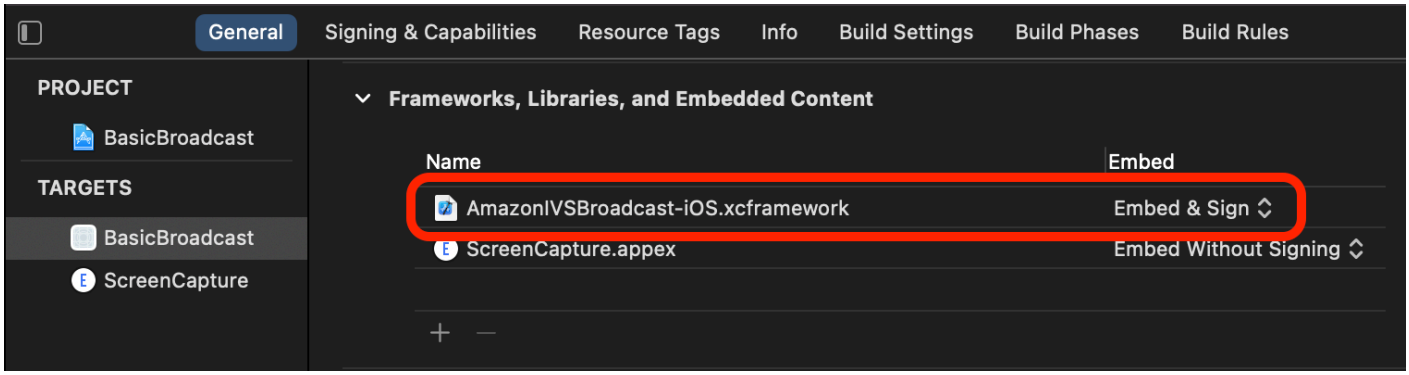
Consigliato: installa Broadcast SDK () CocoaPods

Supponendo che il nome del tuo progetto sia `BasicRealTime`, crea un `Podfile` nella cartella del progetto con il contenuto riportato, quindi esegui `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
```

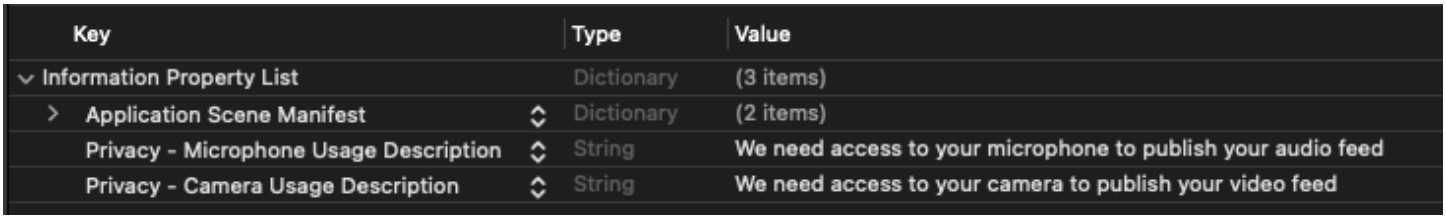
Approccio alternativo: installare manualmente il framework

1. Scaricate la versione più recente da https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Estrai i contenuti dell'archivio. `AmazonIVSBroadcast.xcframework` contiene l'SDK sia per il dispositivo sia per il simulatore.
3. Incorporare `AmazonIVSBroadcast.xcframework` trascinandolo nella sezione Framework, librerie e contenuto incorporato della scheda Generali per il target dell'applicazione:



Per configurare le autorizzazioni

Devi aggiornare `Info.plist` del tuo progetto in modo da aggiungere due nuove voci per `NSCameraUsageDescription` e `NSMicrophoneUsageDescription`. Per i valori, fornisci spiegazioni rivolte all'utente del motivo per cui la tua app richiede l'accesso alla fotocamera e al microfono.



Pubblicazione e sottoscrizione dei video

Vedi i dettagli di seguito per web, Android e iOS.

App

Creazione di un boilerplate HTML

Per prima cosa, creiamo il boilerplate HTML e importiamo la libreria come tag di 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>
```

Accettazione dell'input di token e aggiunta dei pulsanti Unisciti/Abbandona

Qui riempiamo il corpo con i nostri controlli di input. Tali controlli prendono come input il token e configurano i pulsanti Unisciti e Abbandona. In genere le applicazioni richiedono il token dall'API dell'applicazione, ma per questo esempio il token verrà copiato e incollato nell'input del token.

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

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

Aggiunta di elementi del container multimediale

Questi elementi serviranno da supporto ai media per i nostri partecipanti locali e remoti. Aggiungiamo un tag di script per caricare la logica dell'applicazione definita in `app.js`.

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

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

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

Questa operazione completa la pagina HTML che dovrebbe essere visualizzata durante il caricamento di `index.html` in un browser:

Crea app.js

Passiamo alla definizione dei contenuti del file `app.js`. Inizia importando tutte le proprietà richieste dal pacchetto globale dell'SDK:

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

Creazione di variabili dell'applicazione

Stabilisci le variabili che contengono i riferimenti agli elementi HTML dei pulsanti Unisciti e Abbandona e lo stato di archiviazione dell'applicazione:

```
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;
```

Crea joinStage 1: definisci la funzione e convalida l'input

La funzione `joinStage` prende il token di input, crea una connessione alla fase e inizia a pubblicare video e audio recuperati da `getUserMedia`.

Per iniziare, definiamo la funzione e convalidiamo lo stato e l'input del token. Approfondiremo questa funzione nelle prossime sezioni.

```
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
};
```

Crea joinStage 2: ottieni i contenuti multimediali da pubblicare

Ecco i contenuti multimediali che verranno pubblicati nella fase:

```
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]);
```

Crea joinStage 3: definisci la strategia della fase e crea la fase

La strategia della fase è il fulcro della logica decisionale che l'SDK utilizza per decidere cosa pubblicare e quali partecipanti sottoscrivere. Per ulteriori informazioni sullo scopo della funzione, consulta la sezione Strategia.

Questa strategia è semplice. Dopo essere entrati nella fase, pubblica i flussi appena recuperati e sottoscrivi l'audio e i video di ogni partecipante remoto:

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

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

Crea joinStage 4: gestisci gli eventi nella fase ed esegui il rendering dei contenuti multimediali

Le fasi emettono numerosi eventi. Dovremo ascoltare `STAGE_PARTICIPANT_STREAMS_ADDED` e `STAGE_PARTICIPANT_LEFT` per eseguire il rendering e rimuovere contenuti multimediali da e verso la pagina. Una serie più esaustiva di eventi è riportata nella sezione Eventi.

In questo esempio creeremo quattro funzioni di supporto la gestione degli elementi DOM necessari: `setupParticipant`, `teardownParticipant`, `createVideoEl` e `createContainer`.

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

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

```
});

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

    let streamsToDisplay = streams;

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

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

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


// Helper functions for managing DOM

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

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

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

  return videoEl;
}
```

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

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

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

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

  return participantContainer;
}
```

Crea joinStage 5: unisciti alla fase

Completiamo la nostra funzione `joinStage` unendoci finalmente alla fase.

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

Crea leaveStage

Definisci la funzione `leaveStage` che verrà invocata dal pulsante Abbandona.

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

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

Inizializza i gestori di eventi di input

Aggiungeremo un'ultima funzione al file `app.js`. Questa funzione viene richiamata immediatamente quando la pagina viene caricata e stabilisce i gestori di eventi per unirsi e abbandonare la fase.

```
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
```

Esegui l'applicazione e fornisci un token

A questo punto puoi condividere la pagina Web localmente o con altri, apri la pagina e inserisci un token di partecipazione ed entra nella fase.

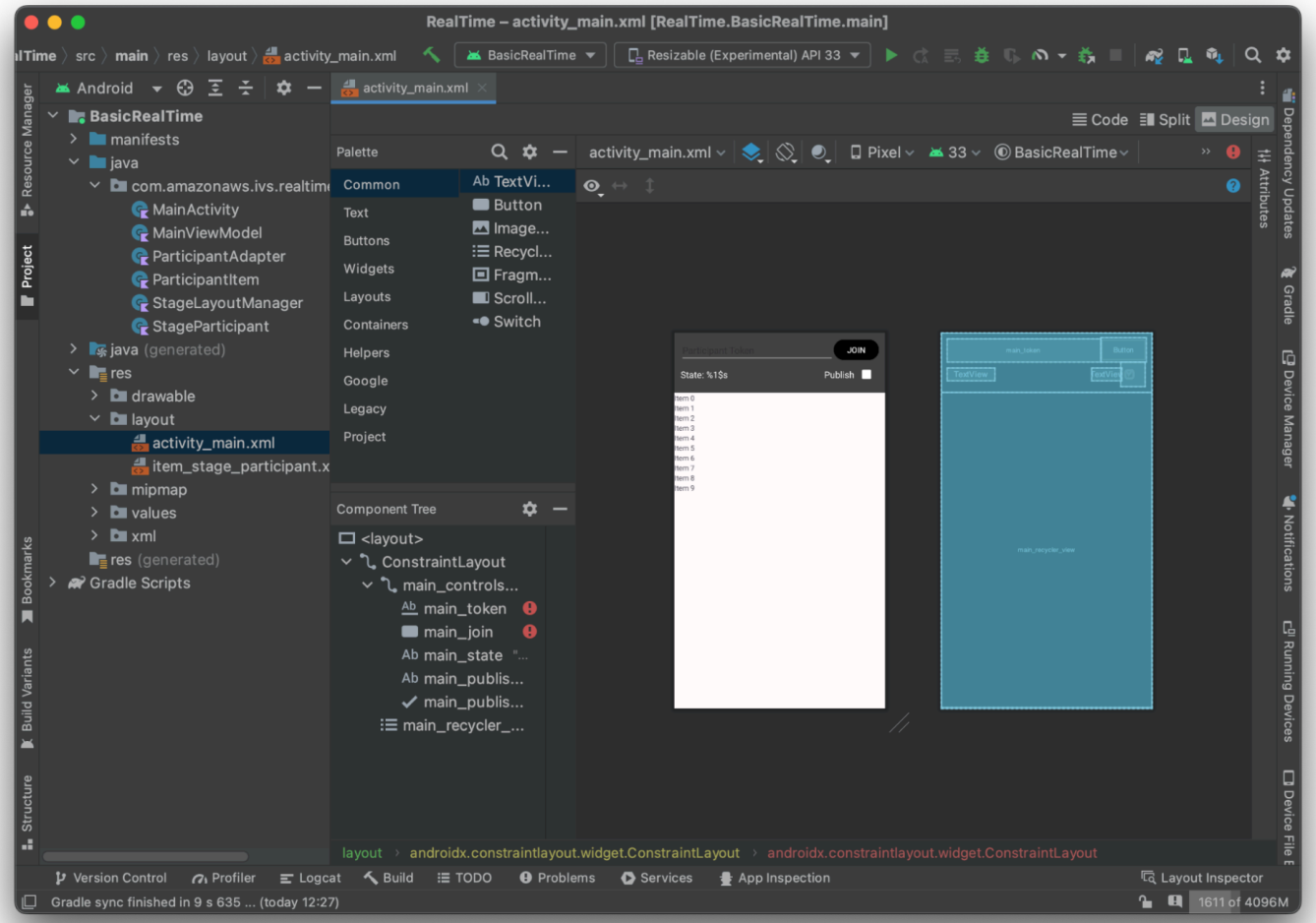
Fasi successive

Per esempi più dettagliati che coinvolgono npm, React e altro, consulta SDK di trasmissione IVS: guida Web (guida per lo streaming in tempo reale).

Android

Creazione delle viste

Iniziamo creando un layout semplice per la nostra app utilizzando il file `activity_main.xml` creato automaticamente. Il layout contiene un `EditText` per aggiungere un token, un `Button` Unisciti, un `TextView` per visualizzare lo stato della fase e un `CheckBox` per attivare la pubblicazione.



Ecco l'XML alla base della vista:

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

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

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

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

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

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

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

            <TextView
```

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

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

        </androidx.constraintlayout.widget.ConstraintLayout>

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

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

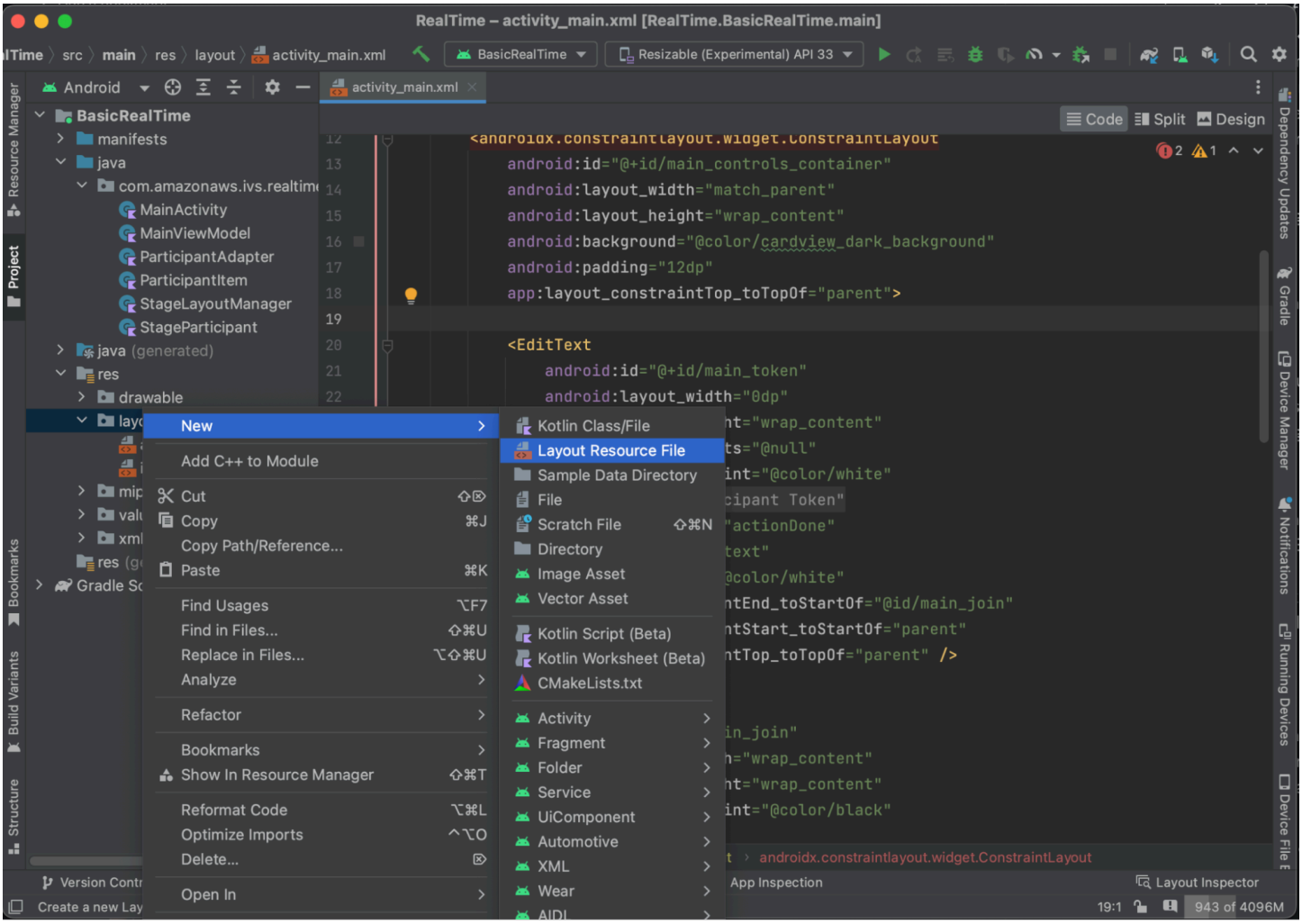
Abbiamo fatto riferimento a un paio di ID di stringa, quindi creeremo il nostro file `strings.xml` per intero:

```
<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>
```

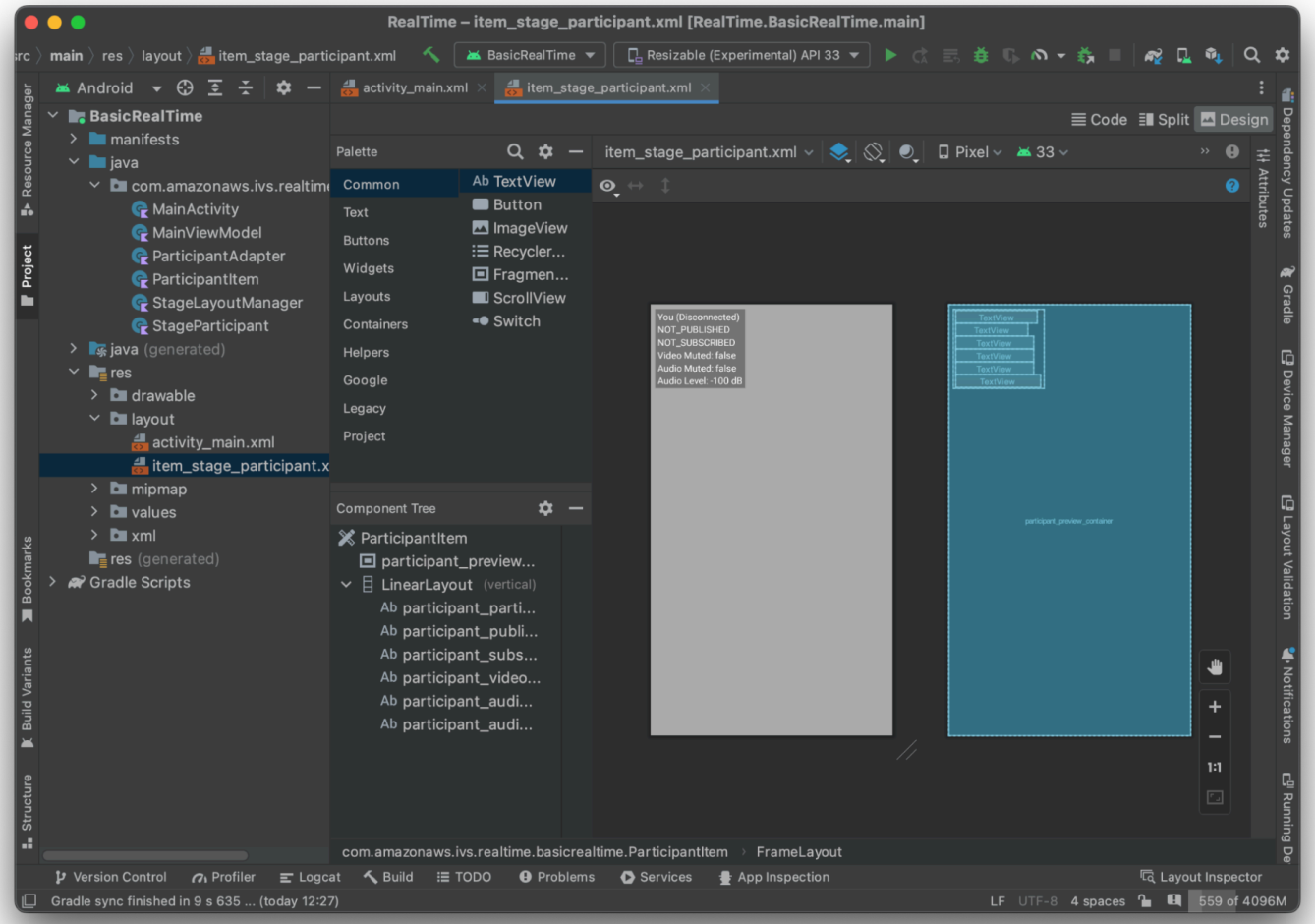
Colleghiamo queste viste nell'XML a `MainActivity.kt`:

```
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)
}
```

Ora creiamo una vista degli elementi per `RecyclerView`. Per fare ciò, fai clic con il pulsante destro del mouse sulla directory `res/layout` e selezionaNuovo > File di risorse di layout. Assegna un nome a questo nuovo file `item_stage_participant.xml`.



Il layout di questo elemento è semplice: contiene una vista per il rendering del flusso video di un partecipante e un elenco di etichette per visualizzare le informazioni sul partecipante:



Ecco il codice 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>
```

Questo file XML inizia una classe che non è stata ancora creata, `ParticipantItem`. Poiché l'XML include lo spazio dei nomi completo, assicurati di aggiornare il file XML con il tuo spazio dei nomi. Creiamo questa classe e configuriamo le viste, ma per ora lasciamola vuota.

Crea una nuova 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)
    }
}
```

Autorizzazioni

Per utilizzare la fotocamera e il microfono, è necessario richiedere le autorizzazioni all'utente. A tal fine seguiamo un flusso di autorizzazioni 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
    }
}
```

Stato dell'app

La nostra applicazione tiene traccia dei partecipanti localmente in a `MainViewModel.kt` e lo stato verrà comunicato all'utente che `MainActivity` utilizza Kotlin. StateFlow

Crea una nuova 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 {

}
```

In `MainActivity.kt` gestiamo il nostro modello di viste:

```
import androidx.activity.viewModels

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

Per usare `AndroidViewModel` e queste estensioni `ViewModel` di Kotlin, dovrai aggiungere quanto segue al file `build.gradle` del modulo:

```
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 Adattatore

Creeremo una semplice sottoclasse `RecyclerView.Adapter` per tenere traccia dei partecipanti e aggiornare `RecyclerView` relativamente agli eventi della fase. Ma prima, abbiamo bisogno di una classe che rappresenti un partecipante. Crea una nuova 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)
            }
        }
}
```

Useremo questa classe nella classe `ParticipantAdapter`che verrà creata successivamente. Iniziamo definendo la classe e creando una variabile per tenere traccia dei partecipanti:

```
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>()
```

Dobbiamo anche definire il nostro `RecyclerView.ViewHolder` prima di implementare il resto delle sostituzioni:

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

In questo modo, possiamo implementare le sostituzioni `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)
```

```
    }
}
```

Infine, aggiungiamo nuovi metodi che chiameremo da `MainViewModel` quando vengono apportate modifiche ai partecipanti. Questi metodi sono operazioni CRUD standard sull'adattatore.

```
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])
    }
}
```

Di nuovo in `MainViewModel` dobbiamo creare e conservare un riferimento a questo adattatore:

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

Stato della fase

Dobbiamo anche tenere traccia di alcuni stati della fase all'interno di `MainViewModel`. Definiamo ora queste proprietà:

```
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>()
```

Per vedere l'anteprima prima di entrare nella fase, creiamo immediatamente un partecipante locale:

```
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)
}
```

Vogliamo assicurarci di cancellare queste risorse quando viene cancellato `ViewModel`. Sostituiamo subito `onCleared()` in modo da non dimenticare di cancellare queste risorse.

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

Ora completiamo la proprietà `streams` locale appena vengono concesse le autorizzazioni, implementando il metodo `permissionsGranted` richiamato in precedenza:

```
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)
    }
}
```

Implementazione dell'SDK della fase

La funzionalità in tempo reale si basa su tre concetti fondamentali: fase, strategia e renderer. L'obiettivo di progettazione è ridurre al minimo la quantità di logica lato client necessaria per creare un prodotto funzionante.

Stage.Strategy

L'implementazione di `Stage.Strategy` è semplice:

```
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
}
```

Per riassumere, eseguiamo la pubblicazione in base allo stato interno di `publishEnabled` e pubblichiamo i flussi raccolti in precedenza. Infine, per questo esempio, sottoscriviamo sempre gli altri partecipanti, ricevendo sia il loro audio che i loro video.

StageRenderer

Anche l'implementazione di `StageRenderer` è abbastanza semplice, sebbene dato il numero di funzioni contenga un po' più di codice. L'approccio generale in questo renderer è quello di aggiornare `ParticipantAdapter` quando l'SDK notifica una modifica a un partecipante. Ci sono alcuni scenari in cui gestiamo i partecipanti locali in modo diverso, perché abbiamo deciso di gestirli noi stessi in modo che possano vedere l'anteprima della fotocamera prima di partecipare.

```
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) {}
}
```

Implementazione di una personalizzazione RecyclerView LayoutManager

La creazione di un layout con un numero diverso di partecipanti può essere complessa. Vuoi che occupino l'intera cornice della vista principale, ma non vuoi gestire singolarmente la configurazione di ogni partecipante. Per semplificare questa operazione, descriveremo l'implementazione di un `RecyclerView.LayoutManager`.

Crea un'altra nuova classe, `StageLayoutManager`, che dovrebbe estendere `GridLayoutManager`. Questa classe è progettata per calcolare il layout per ogni partecipante in base al numero di partecipanti in un layout di riga/colonna basato sul flusso. Ogni riga ha la stessa altezza delle altre, ma le colonne possono avere larghezze diverse a seconda della riga. Vedi il commento sul codice sopra la variabile `layouts` per una descrizione di come personalizzare questo comportamento.

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

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

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

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

```
            listOf(1), // 1 row, full width
            // 2 participants
            listOf(1, 1), // 2 rows, all columns are full width
            // 3 participants
            listOf(1, 2), // 2 rows, first row's column is full width then 2nd row's
columns are 1/2 width
            // 4 participants
            listOf(2, 2), // 2 rows, all columns are 1/2 width
            // 5 participants
            listOf(1, 2, 2), // 3 rows, first row's column is full width, 2nd and 3rd
row's columns are 1/2 width
            // 6 participants
            listOf(2, 2, 2), // 3 rows, all column are 1/2 width
            // 7 participants
            listOf(2, 2, 3), // 3 rows, 1st and 2nd row's columns are 1/2 width, 3rd
row's columns are 1/3rd width
            // 8 participants
            listOf(2, 3, 3),
            // 9 participants
            listOf(3, 3, 3),
            // 10 participants
            listOf(2, 3, 2, 3),
            // 11 participants
            listOf(2, 3, 3, 3),
            // 12 participants
            listOf(3, 3, 3, 3),
        )
    }

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

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

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

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

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

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

Di nuovo in `MainActivity.kt`, dobbiamo impostare l'adattatore e il gestore del layout per il `RecyclerView`:

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

Aggancio delle operazioni dell'interfaccia utente

Stiamo per finire; ci sono solo alcune operazioni dell'interfaccia utente che dobbiamo agganciare.

Per prima cosa dobbiamo far sì che `MainActivity` osservi le modifiche `StateFlow` da `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)
        }
    }
}
```

Successivamente dobbiamo aggiungere gli ascoltatori al nostro pulsante Unisciti e alla casella di controllo Pubblica:

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

Entrambe le funzionalità di chiamata sopra riportate nel `MainViewModel`, che implementiamo ora:

```
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
}
```

Rendering dei partecipanti

Infine, dobbiamo eseguire il rendering dei dati che riceviamo dall'SDK sull'elemento del partecipante che abbiamo creato in precedenza. Abbiamo già la logica `RecyclerView` finita, quindi dobbiamo solo implementare l'API `bind` in `ParticipantItem`.

Inizieremo aggiungendo la funzione vuota e poi la esamineremo dettagliatamente:

```
fun bind(participant: StageParticipant) {

}
```

Per prima cosa gestiremo lo stato di facilità, l'ID partecipante, lo stato di pubblicazione e lo stato di sottoscrizione. Per questi, ci limitiamo ad aggiornare direttamente il `TextViews`:

```
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
```

Successivamente aggiorneremo gli stati di disattivazione audio e video. Per ottenere lo stato di audio disattivato, dobbiamo trovare `ImageDevice` e `AudioDevice` dall'array dei flussi. Per ottimizzare le prestazioni, ricordiamo gli ID degli ultimi dispositivi collegati.

```
// 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"
}
```

Infine vogliamo eseguire il rendering di un'anteprima per `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
```

E mostriamo le statistiche audio di `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
```

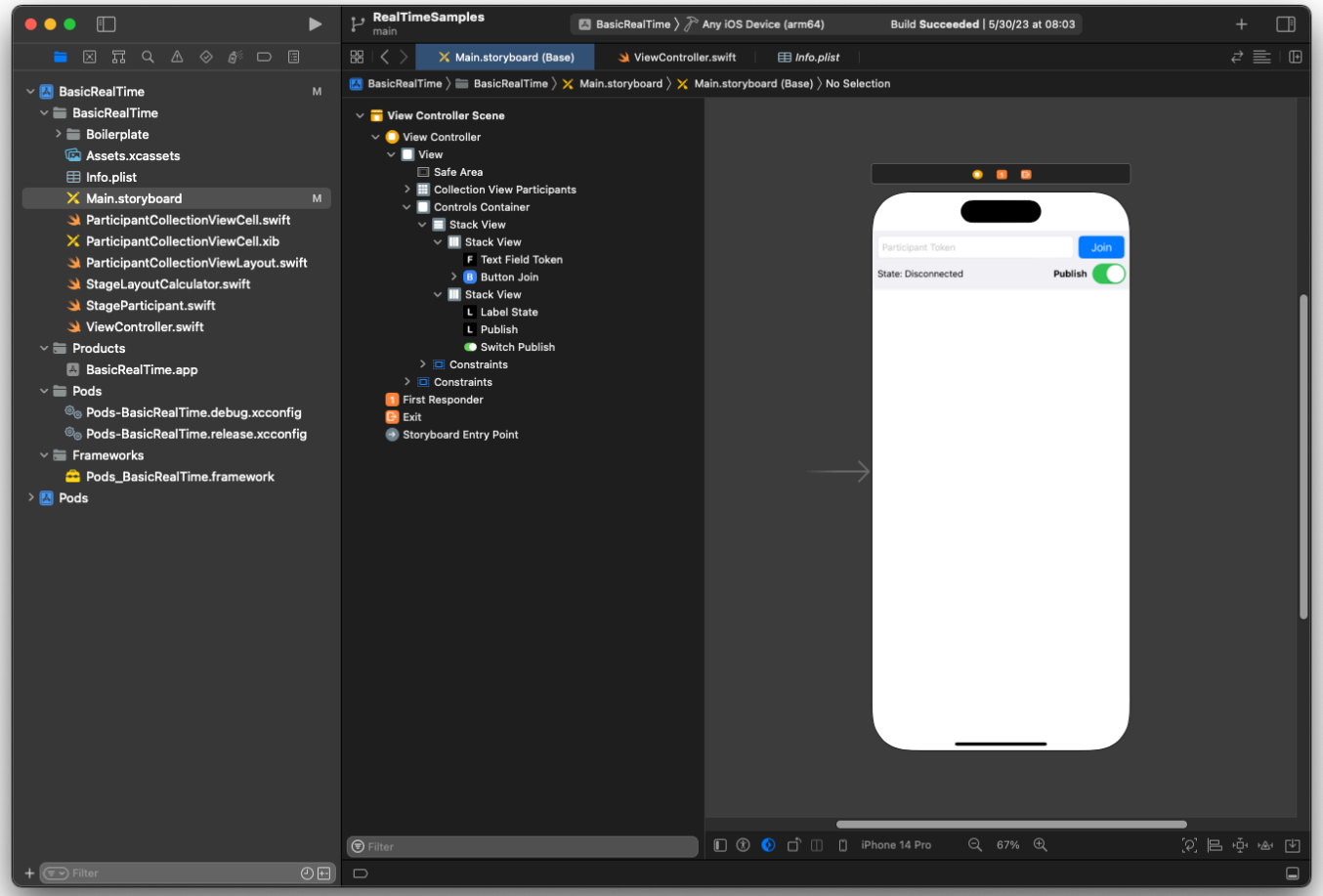
iOS

Creazione delle viste

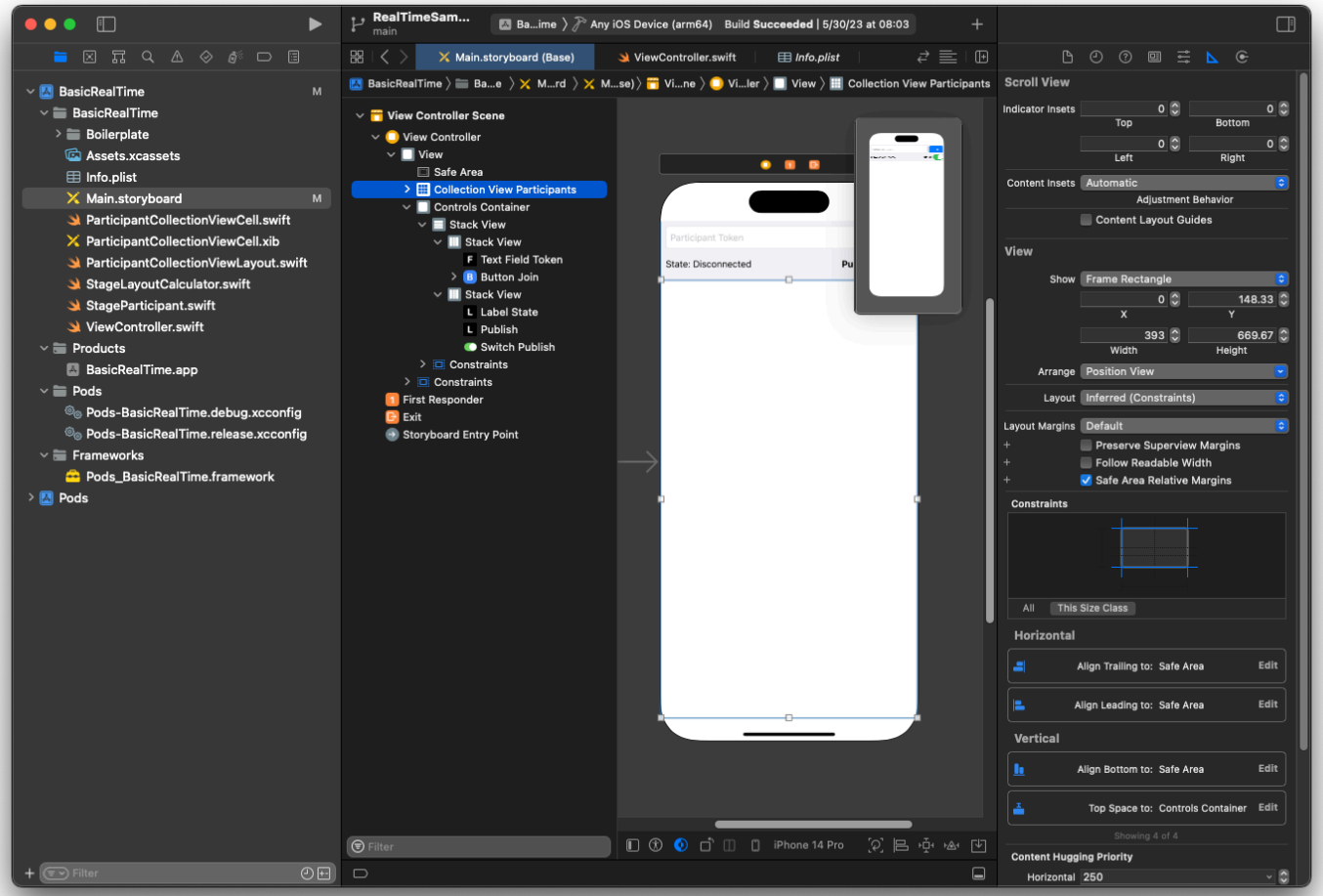
Iniziamo usando il file `ViewController.swift` creato automaticamente per importare `AmazonIVSBroadcast` e poi aggiungiamo `@IBOutlets` per collegare:

```
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!
```

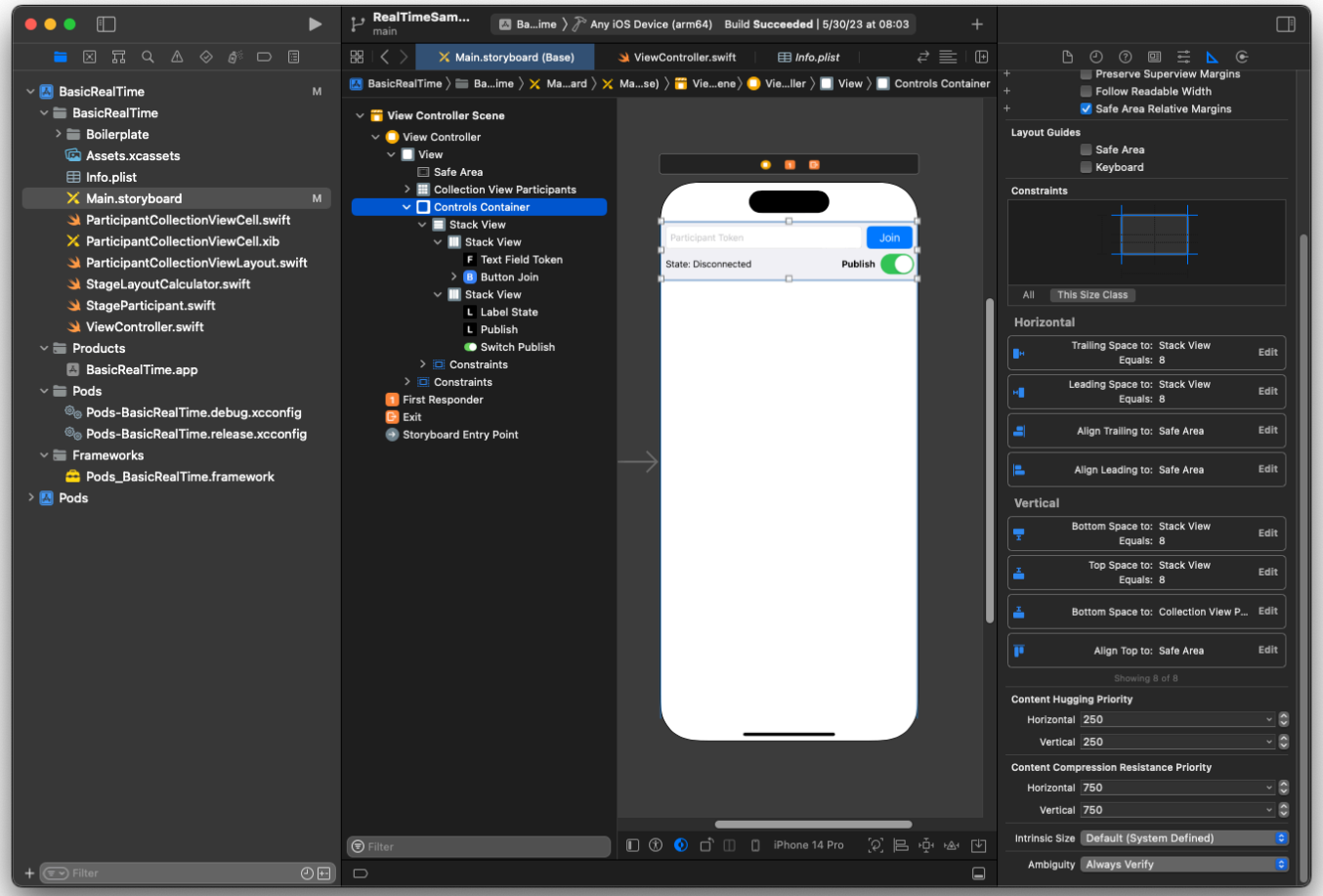
A questo punto creiamo quelle viste e le colleghiamo in `Main.storyboard`. Ecco la struttura delle viste che useremo:



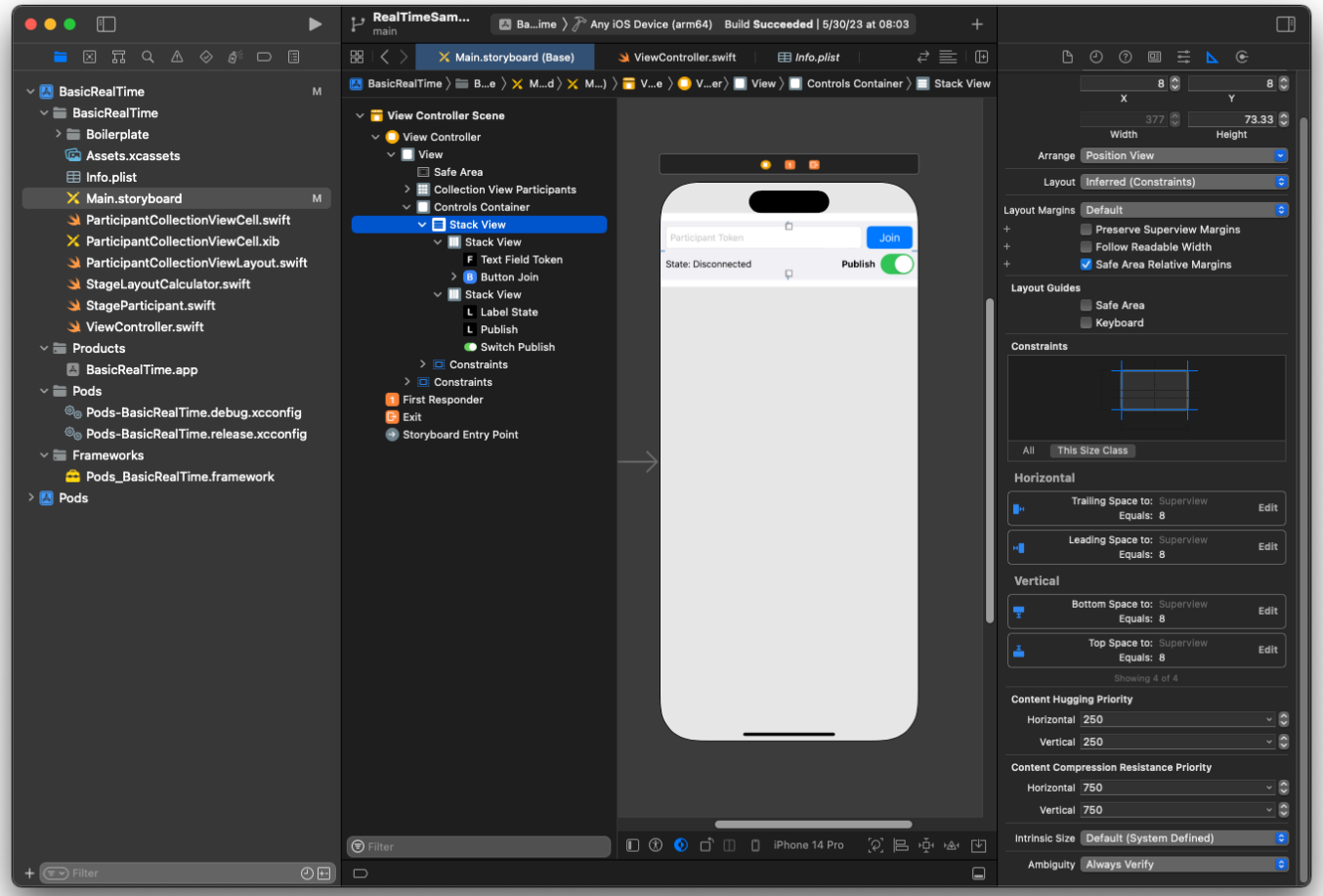
Per la AutoLayout configurazione, dobbiamo personalizzare tre viste. La prima vista è Partecipanti della vista Raccolta (`UICollectionView`). Collegato Iniziale, Finale e In basso a Area sicura. Collegato anche In alto a Container controlli.



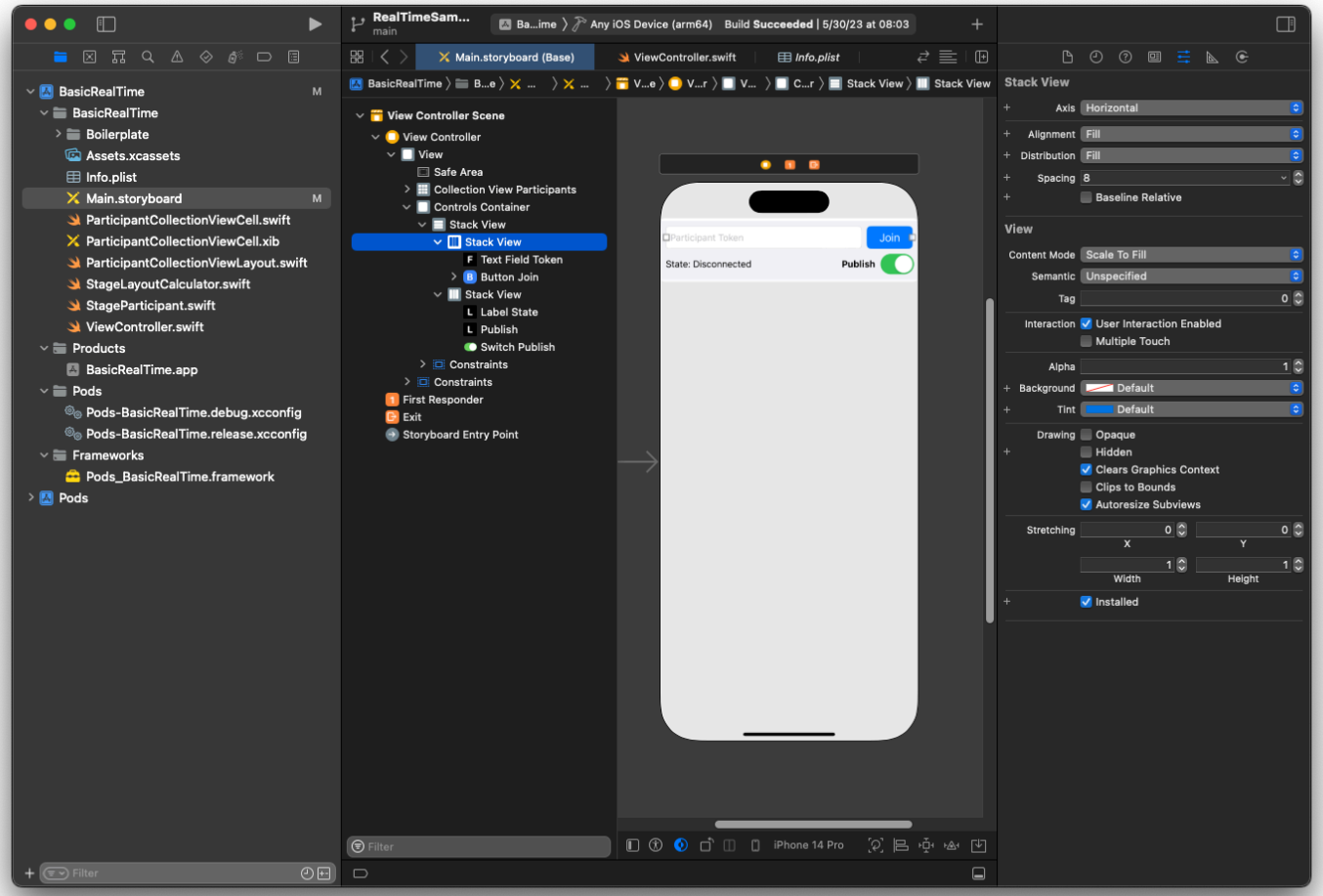
La seconda vista è Container controlli. Collegato Iniziale, Finale e In alto a Area sicura.



La terza e ultima vista è Vista Stack verticale. Collegato In alto, Iniziale, Finale e In basso a Supervista. Per lo stile, imposta la spaziatura su 8 anziché su 0.



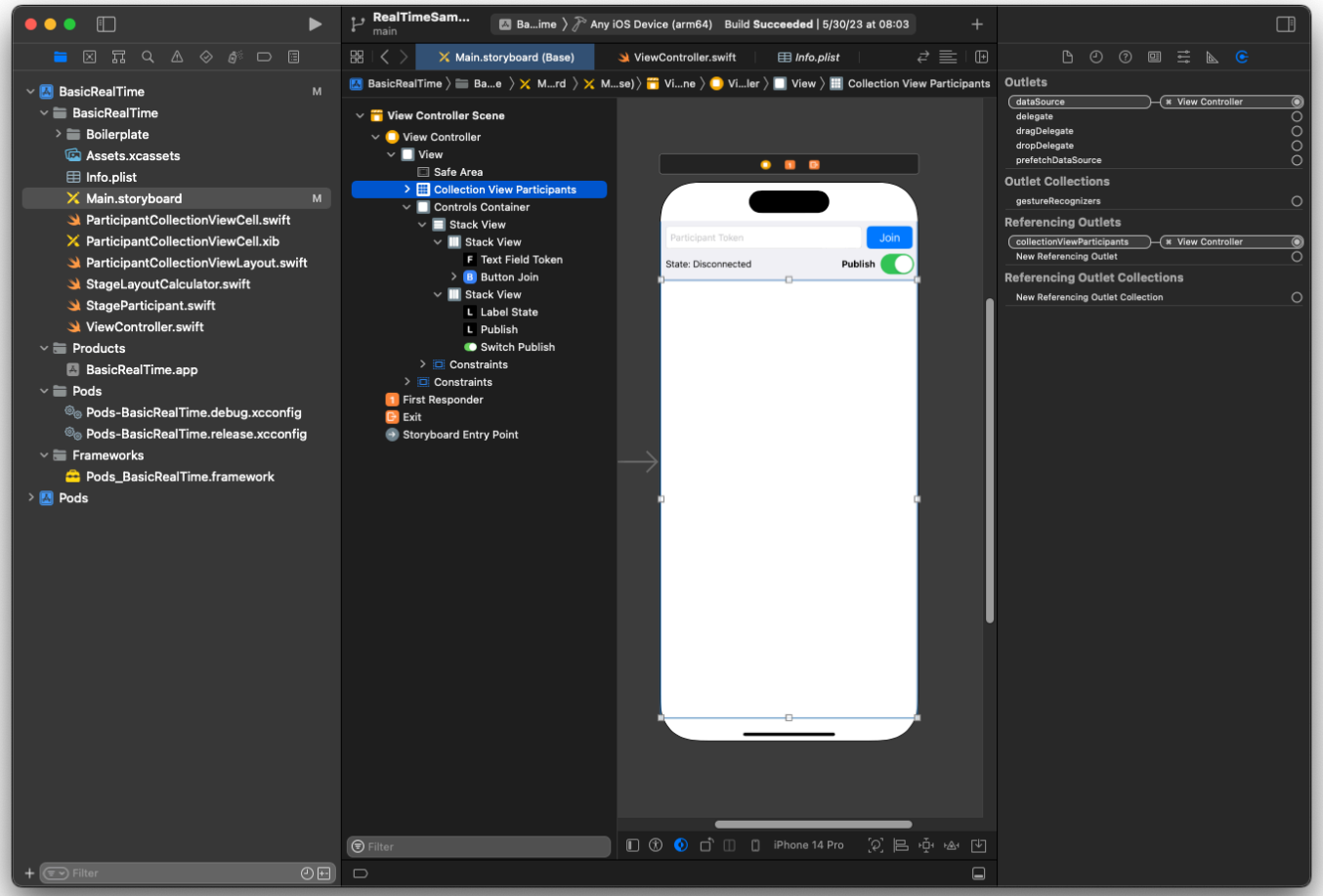
L'interfaccia utente StackViews gestirà il layout delle viste rimanenti. Per tutte e tre le interfacce utente StackViews, usa Fill come allineamento e distribuzione.



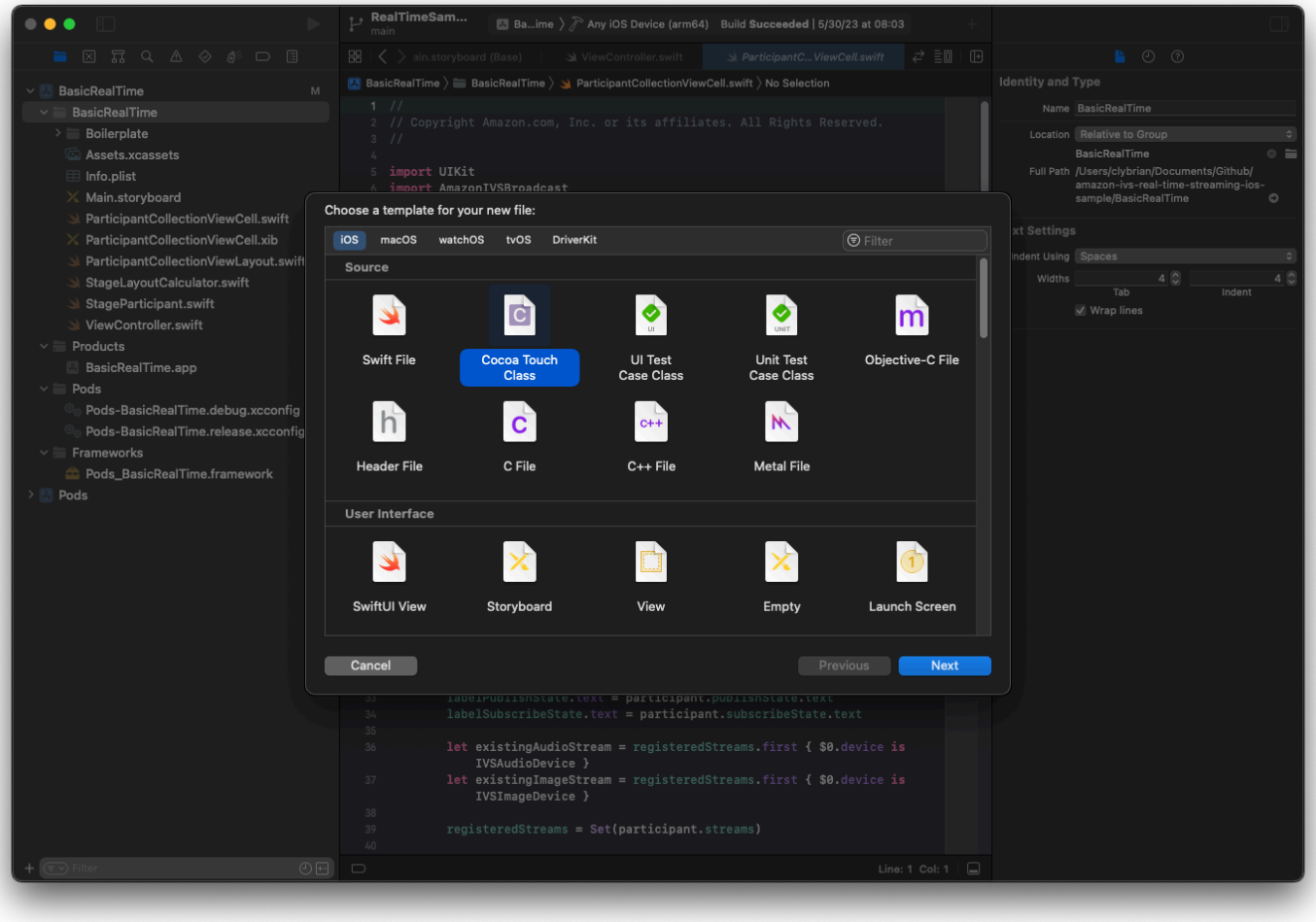
Infine, colleghiamo queste viste a `ViewController`. Dall'alto, mappa le seguenti viste:

- Il campo di testo Unisciti si collega a `textFieldToken`.
- Il pulsante Unisciti si collega a `buttonJoin`.
- Lo stato dell'etichetta si collega a `labelState`.
- Cambia pubblicazione si collega a `switchPublish`.
- Partecipanti della vista Raccolta si collega a `collectionViewParticipants`.

Usa questo tempo anche per impostare `dataSource` dell'elemento Partecipanti della vista Raccolta al `ViewController` di proprietà:



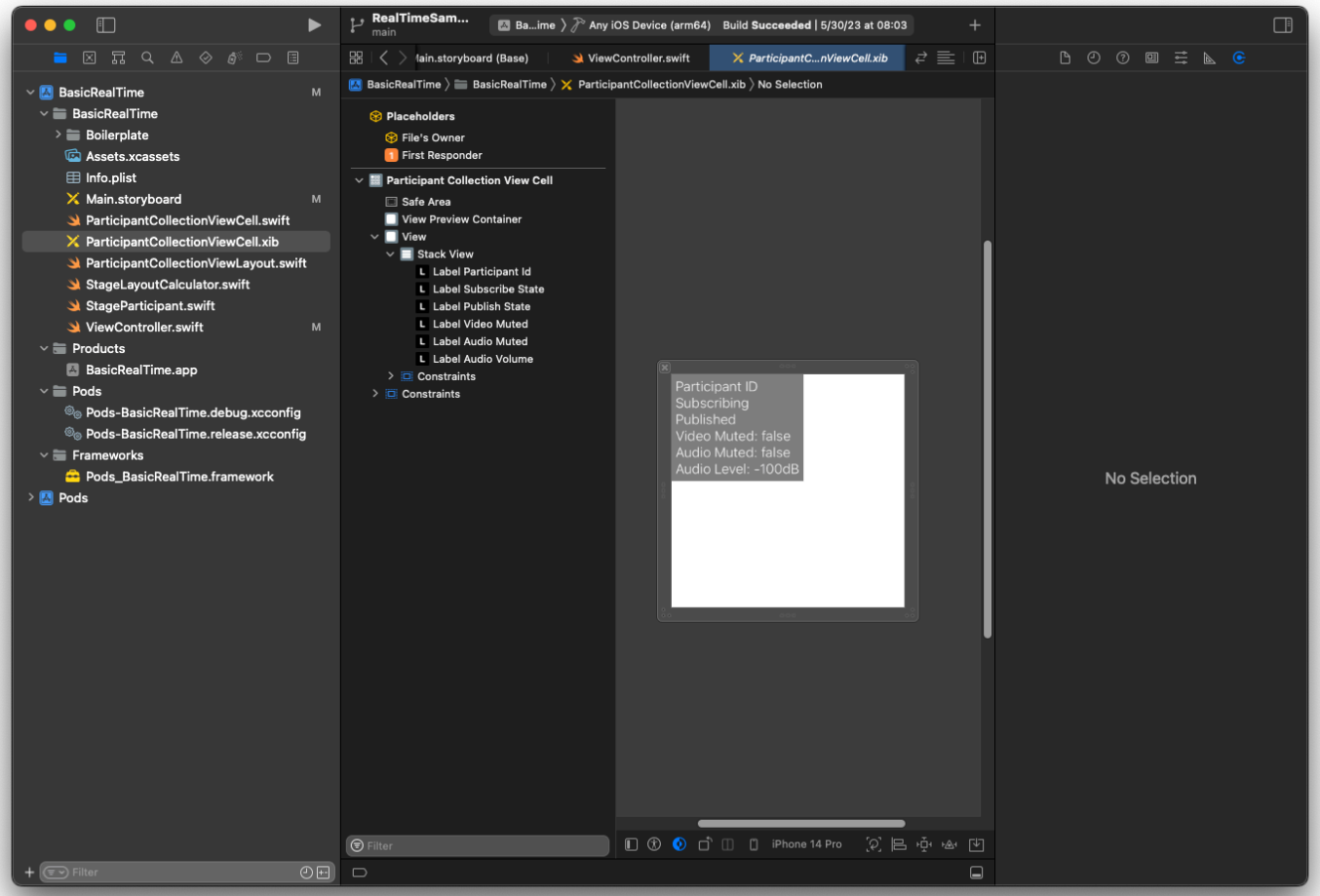
A questo punto creiamo la sottoclasse `UICollectionViewCell` in cui eseguire il rendering dei partecipanti. Inizia creando un nuovo file Classe Cocoa Touch:



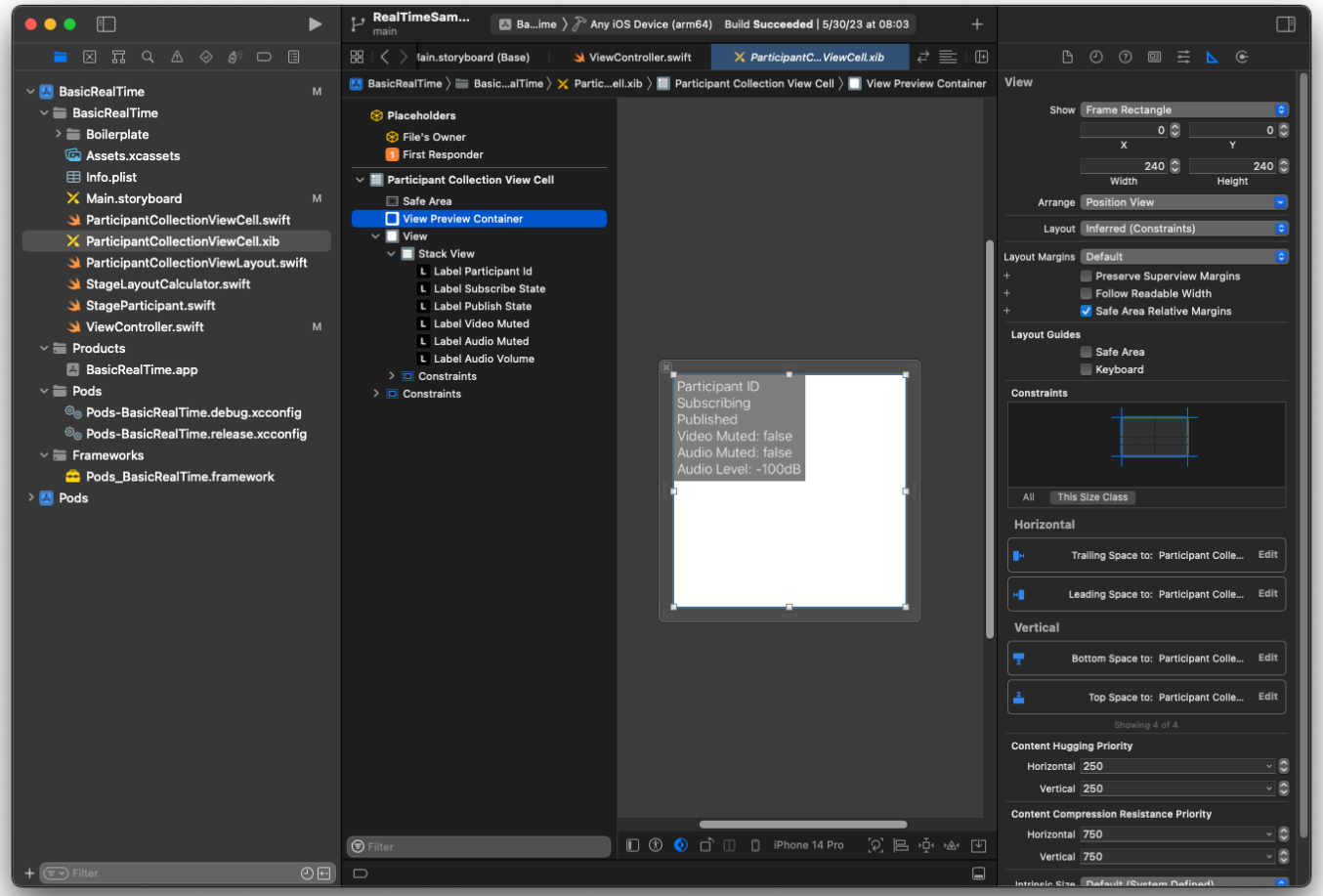
Denominalo `ParticipantUICollectionViewCell` e rendilo una sottoclasse di `UICollectionViewCell` in Swift. Ricominciamo dal file Swift, creando il `@IBOutlets`per collegare:

```
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!
```

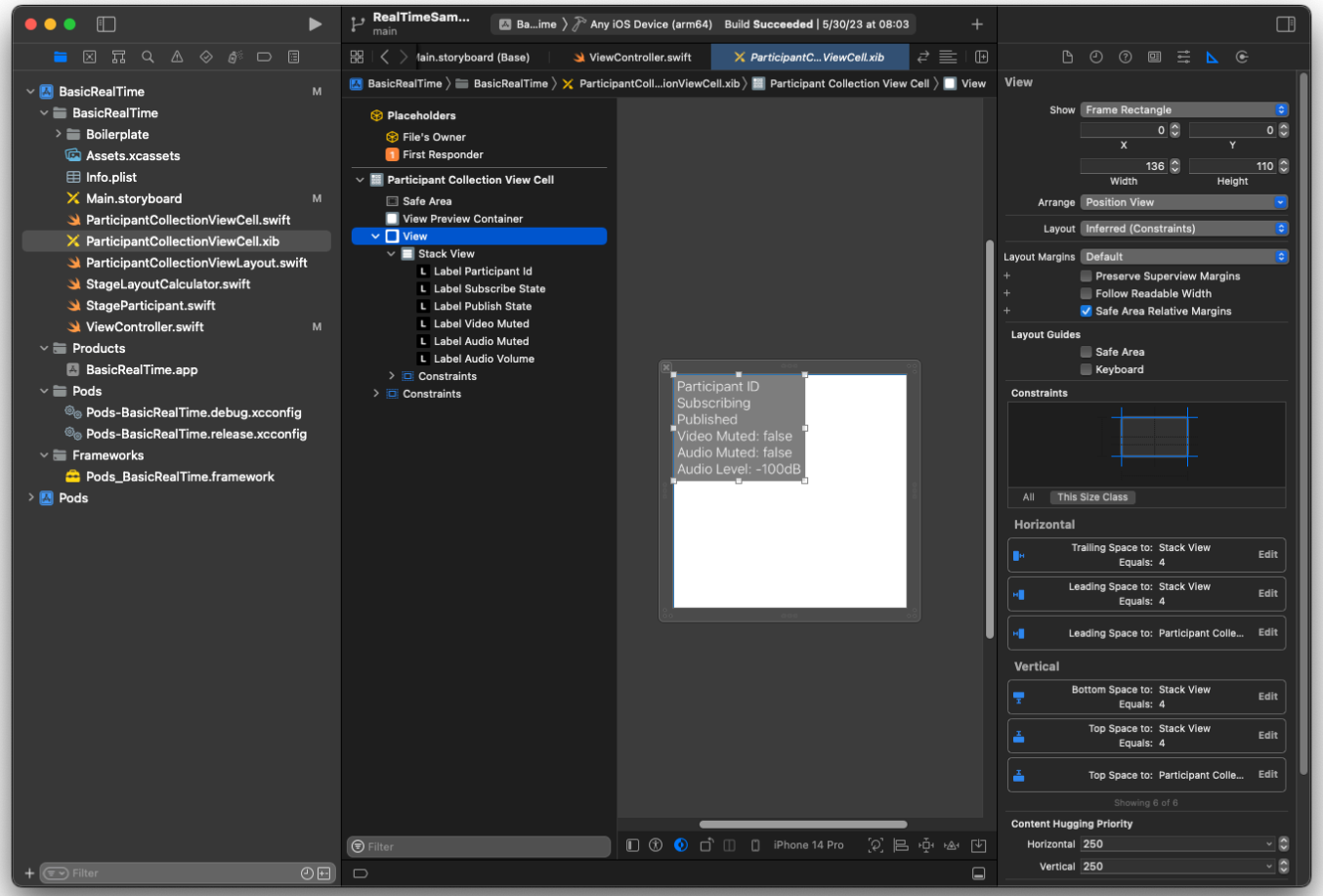
Nel file XIB associato, crea questa gerarchia di viste:



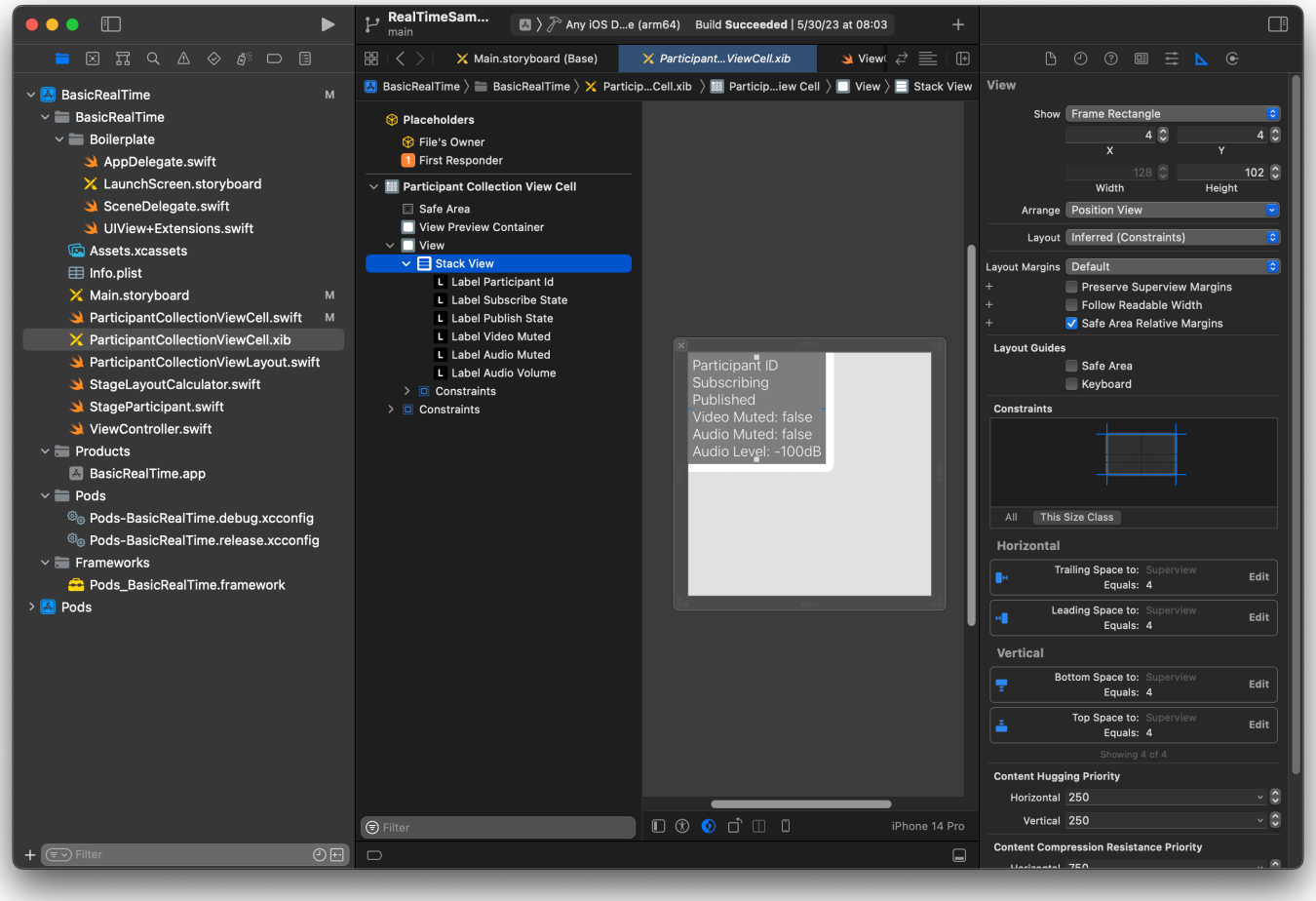
Infatti AutoLayout, modificheremo nuovamente tre viste. La prima vista è Container di anteprima vista. Imposta Finale, Iniziale, In alto e In basso su Cella vista raccolta partecipanti.



La seconda vista è Vista. Imposta Iniziale e In alto su Cella vista raccolta partecipanti e modifica il valore su 4.



La terza vista è Vista stack. Imposta Finale, Iniziale, In alto e In basso su Supervista, quindi modifica il valore su 4.



Autorizzazioni e timer di inattività

Tornando a `ViewController`, disabiliteremo il timer di inattività del sistema per impedire al dispositivo di andare in modalità standby mentre la nostra applicazione è in uso:

```
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
}
```

Successivamente richiederemo al sistema le autorizzazioni per fotocamera e microfono:

```
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)
    }
}
```

Stato dell'app

Dobbiamo configurare `collectionViewParticipants` con il file di layout creato in precedenza:

```
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")
}
```

Per rappresentare ogni partecipante, creiamo una semplice struttura chiamata `StageParticipant`. Questa può essere inclusa nel file `ViewController.swift` oppure è possibile creare un nuovo file.

```
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
    }
}
```

Per tenere traccia di questi partecipanti, ne conserviamo un array come proprietà privata nel nostro `ViewController`:

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

Questa proprietà verrà utilizzata per alimentare `UICollectionViewDataSource` che era collegato allo storyboard precedente:

```
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'")
        }
    }

}
```

Per vedere l'anteprima prima di entrare nella fase, creiamo immediatamente un partecipante locale:

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

Ciò comporta il rendering di una cella partecipante immediatamente dopo l'esecuzione dell'app, che rappresenta il partecipante locale.

Gli utenti vogliono essere in grado di vedere se stessi prima di unirsi a una fase, quindi ora implementeremo il metodo `setupLocalUser()` che viene chiamato prima dal codice di gestione delle autorizzazioni. Memorizziamo il riferimento della fotocamera e del microfono come oggetti `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)
}
```

Qui abbiamo trovato la fotocamera e il microfono del dispositivo tramite l'SDK e li abbiamo archiviati nel nostro oggetto `streams` locale, quindi abbiamo assegnato l'array `streams` del primo partecipante (il partecipante locale che abbiamo creato in precedenza) al nostro `streams`. Finalmente richiamiamo `participantsChanged` con un `index` pari a 0 e `changeType` pari a `updated`. Questa funzione è una funzione helper per aggiornare il nostro `UICollectionView` con simpatiche animazioni. Ecco come appare:

```
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)])
    }
}
```

Non preoccuparti ancora di `cell.set`, ci arriveremo più tardi, ma è lì che eseguiremo il rendering del contenuto della cella in base al partecipante.

`ChangeType` è un semplice enum:

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

```
}
```

Infine, vogliamo verificare se la fase è collegata. Per tenerne traccia, usiamo un semplice `bool`, che aggiornerà automaticamente la nostra interfaccia utente quando si aggiornata da sola.

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

Implementazione dell'SDK della fase

La funzionalità in tempo reale si basa su tre concetti fondamentali: fase, strategia e renderer. L'obiettivo di progettazione è ridurre al minimo la quantità di logica lato client necessaria per creare un prodotto funzionante.

IVS StageStrategy

L'implementazione di `IVSStageStrategy` è semplice:

```
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
    }
}
```

Per riassumere, eseguiamo la pubblicazione solo se l'opzione di pubblicazione è in posizione "on" e, se pubblichiamo, pubblicheremo i flussi raccolti in precedenza. Infine, per questo esempio, sottoscriviamo sempre gli altri partecipanti, ricevendo sia il loro audio che i loro video.

IVS StageRenderer

Anche l'implementazione di `IVSStageRenderer` è abbastanza semplice, sebbene dato il numero di funzioni contenga un po' più di codice. L'approccio generale in questo renderer è quello di aggiornare l'array `participants` quando l'SDK notifica una modifica a un partecipante. Ci sono alcuni scenari in cui gestiamo i partecipanti locali in modo diverso, perché abbiamo deciso di gestirli noi stessi in modo che possano vedere l'anteprima della fotocamera prima di partecipare.

```
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)
    }
}
```

Questo codice utilizza un'estensione per convertire lo stato della connessione in testo intuitivo per l'utente:

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

Implementazione di un'interfaccia utente personalizzata CollectionViewLayout

La creazione di un layout con un numero diverso di partecipanti può essere complessa. Vuoi che occupino l'intera cornice della vista principale, ma non vuoi gestire singolarmente la configurazione di ogni partecipante. Per semplificare questa operazione, descriveremo l'implementazione di un `UICollectionViewLayout`.

Crea un altra nuovo file, `ParticipantCollectionViewLayout.swift`, che dovrebbe estendere `UICollectionViewLayout`. Questa classe userà un'altra classe chiamata `StageLayoutCalculator`, di cui parleremo presto. La classe riceve i valori di frame calcolati per ogni partecipante e quindi genera gli oggetti `UICollectionViewLayoutAttributes` necessari.

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

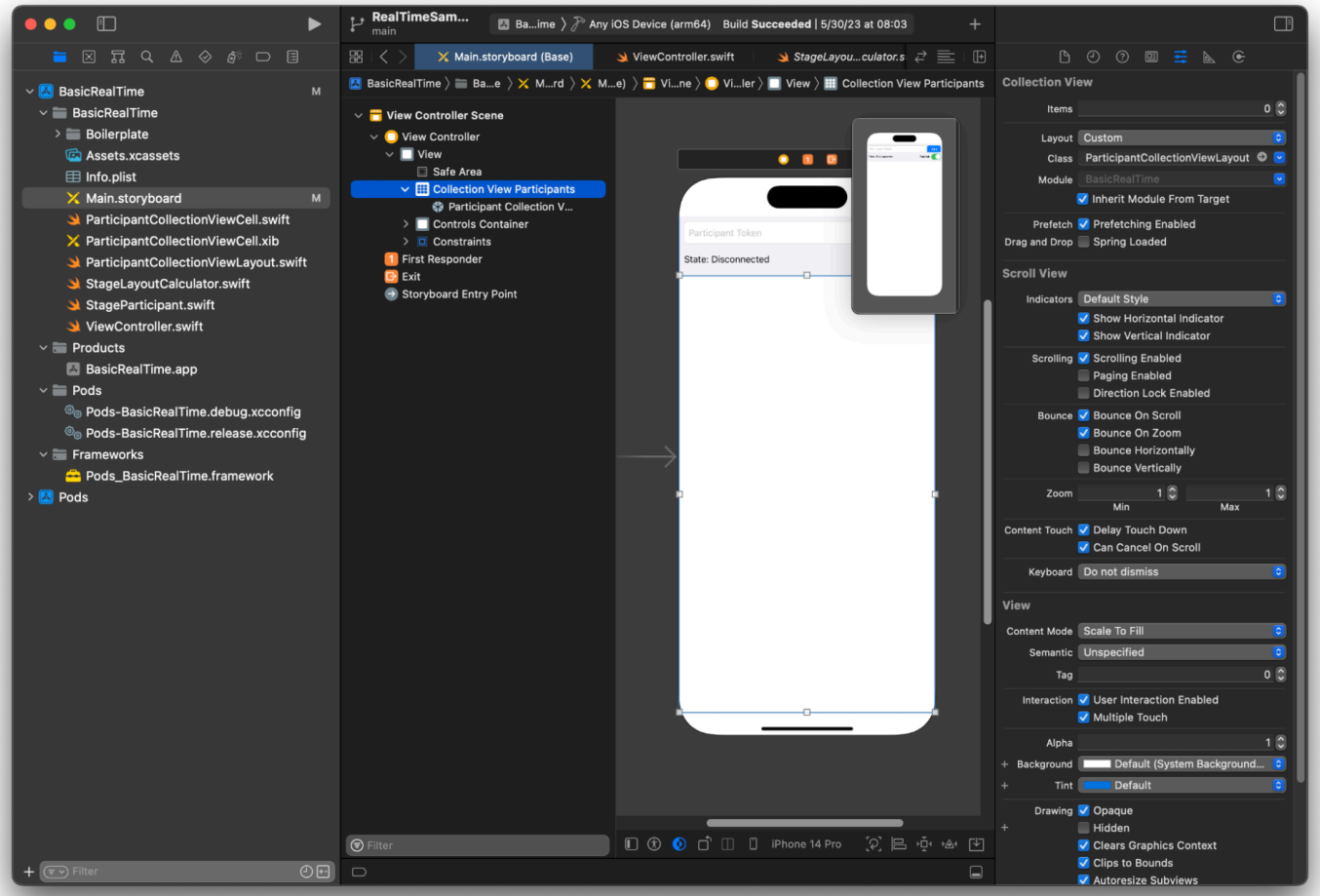
Più importante è la classe `StageLayoutCalculator.swift`. Questa classe è progettata per calcolare i frame per ogni partecipante in base al numero di partecipanti in un layout di riga/colonna basato sul flusso. Ogni riga ha la stessa altezza delle altre, ma le colonne possono avere larghezze diverse a seconda della riga. Vedi il commento sul codice sopra la variabile `layouts` per una descrizione di come personalizzare questo comportamento.

```
import Foundation
import UIKit

class StageLayoutCalculator {

    /// This 2D array contains the description of how the grid of participants should
 be rendered
    /// The index of the 1st dimension is the number of participants needed to active
 that configuration
    /// Meaning if there is 1 participant, index 0 will be used. If there are 5
 participants, index 4 will be used.
    ///
    /// The 2nd dimension is a description of the layout. The length of the array is
 the number of rows that
    /// will exist, and then each number within that array is the number of columns in
 each row.
    ///
    /// See the code comments next to each index for concrete examples.
    ///
    /// This can be customized to fit any layout configuration needed.
    private let layouts: [[Int]] = [
        // 1 participant
        [ 1 ], // 1 row, full width
        // 2 participants
        [ 1, 1 ], // 2 rows, all columns are full width
        // 3 participants
        [ 1, 2 ], // 2 rows, first row's column is full width then 2nd row's columns
 are 1/2 width
        // 4 participants
        [ 2, 2 ], // 2 rows, all columns are 1/2 width
        // 5 participants
        [ 1, 2, 2 ], // 3 rows, first row's column is full width, 2nd and 3rd row's
 columns are 1/2 width
        // 6 participants
        [ 2, 2, 2 ], // 3 rows, all column are 1/2 width
        // 7 participants
        [ 2, 2, 3 ], // 3 rows, 1st and 2nd row's columns are 1/2 width, 3rd row's
 columns are 1/3rd width
        // 8 participants
        [ 2, 3, 3 ],
        // 9 participants
        [ 3, 3, 3 ],
        // 10 participants
```

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

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

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

        let halfPadding = padding / 2.0

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

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

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

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

}
```

Di nuovo in `Main.storyboard`, assicurati di impostare la classe di layout per `UICollectionView` sulla classe che abbiamo appena creato:



Aggancio delle operazioni dell'interfaccia utente

Stiamo per finire, dobbiamo solo creare qualche `IBActions`.

Per prima cosa gestiremo il pulsante Unisciti. Questo pulsante risponde in modo diverso in base al valore di `connectingOrConnected`. Quando è già connesso, abbandona semplicemente la fase. Se è disconnesso, legge il testo dal token `UITextField` e crea un nuovo `IVSStage` con quel testo. Quindi aggiungiamo il nostro `ViewController` come `strategy`,`errorDelegate` e renderer per `IVSStage`, infine ci uniamo alla fase in modo asincrono.

```
@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'altra operazione dell'interfaccia utente che dobbiamo agganciare è il cambio di pubblicazione:

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

Rendering dei partecipanti

Infine, dobbiamo eseguire il rendering dei dati che riceviamo dall'SDK sulla cella del partecipante che abbiamo creato in precedenza. Abbiamo già la logica `UICollectionView` finita, quindi dobbiamo solo implementare l'API `set` in `ParticipantCollectionViewCell.swift`.

Inizieremo aggiungendo la funzione `empty` e poi la esamineremo dettagliatamente:

```
func set(participant: StageParticipant) {

}
```

Per prima cosa gestiremo lo stato di facilità, l'ID partecipante, lo stato di pubblicazione e lo stato di sottoscrizione. Per questi, ci limitiamo ad aggiornare direttamente il `UILabels`:

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

Le proprietà testuali delle enumerazioni di pubblicazione e sottoscrizione provengono da estensioni locali:

```
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()
        }
    }
}
```

Successivamente aggiorneremo gli stati di disattivazione audio e video. Per ottenere lo stato di audio disattivato, dobbiamo trovare `IVSImageDevice` e `IVSAudioDevice` dall'array `streams`. Per ottimizzare le prestazioni, ricorderemo gli ultimi dispositivi collegati.

```
// 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)"
```

Infine vogliamo eseguire il rendering di un'anteprima per `imageDevice` e visualizzare le statistiche audio dal `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"
}
```

L'ultima funzione che dobbiamo creare è `updatePreview()`, che aggiunge un'anteprima del partecipante alla nostra vista:

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

```
    }
}
```

Quanto sopra utilizza una funzione helper su `UIView` per semplificare l'integrazione delle viste secondarie:

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

Monitoraggio dello streaming in tempo reale di Amazon IVS

Che cos'è una sessione di fase?

Una sessione di fase inizia quando il primo partecipante si unisce a una fase e termina pochi minuti dopo che l'ultimo ha smesso di pubblicare nella fase. Le sessioni di fase aiutano a eseguire il debug delle fasi di lunga durata separando eventi e partecipanti in sessioni di breve durata.

Visualizzazione delle sessioni e dei partecipanti alla fase

Istruzioni per la console

1. Aprire la Console Amazon IVS.

   L'accesso alla console Amazon IVS è possibile anche dalla Console di gestione AWS.
2. Nel riquadro di navigazione, scegli Fase. Se il riquadro di navigazione è compresso, aprirlo prima scegliendo l'icona a hamburger.
3. Scegli la fase per accedere alla rispettiva pagina dei dettagli.
4. Scorri la pagina verso il basso fino a visualizzare la sezione Sessioni di fase, quindi seleziona una sessione di fase per visualizzare la rispettiva pagina dei dettagli.
5. Per visualizzare i partecipanti alla sessione, scorri verso il basso fino a visualizzare la sezione Partecipanti, quindi seleziona un partecipante per visualizzare la relativa pagina dei dettagli, inclusi i grafici per i parametri Amazon CloudWatch.

Visualizzazione degli eventi per un partecipante

Gli eventi vengono inviati quando lo stato di un partecipante in una fase cambia, ad esempio se si unisce a una fase o si verifica un errore durante il tentativo di pubblicazione in una fase. Non tutti gli errori causano eventi; ad esempio, gli errori di rete lato client e gli errori di firma dei token non vengono inviati come eventi. Per gestire questi errori nella tua applicazione client, usa gli SDK di trasmissione IVS.

Istruzioni per la console

1. Accedi alla pagina dei dettagli del partecipante come indicato sopra.

2. Scorri verso il basso fino a visualizzare la sezione Eventi. Viene visualizzato un elenco ordinato degli eventi dei partecipanti. Consulta la sezione Utilizzo di Amazon EventBridge con Amazon IVS per dettagli sugli eventi che vengono emessi per i partecipanti.

Istruzioni per la CLI

L'accesso agli eventi delle sessioni di fase con AWS CLI è un'opzione avanzata e richiede prima il download e la configurazione della CLI sul computer in uso. Per maggiori dettagli, consulta la Guida per l'utente dell'interfaccia a riga di comando di AWS.

1. Elenca le sessioni di fase per trovare una sessione di fase:

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

2. Elenca i partecipanti a una sessione di fase per trovare un partecipante:

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

3. Elenca gli eventi per una sessione di fase e per il partecipante:

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

Di seguito è riportata una risposta di esempio alla chiamata `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"
        }
    ]
}
```

Accesso ai parametri di CloudWatch

Affinché i parametri di CloudWatch siano disponibili, sono necessarie le seguenti versioni SDK di trasmissione IVS: Web 1.5.0 o successive, Android 1.12.0 o successive o iOS 1.12.0 o successive.

Istruzioni per la console CloudWatch

1. Aprire la console CloudWatch all'indirizzo https://console.aws.amazon.com/cloudwatch/.
2. Nella navigazione laterale, espandere il menu a discesa Metrics (Parametri), quindi selezionare All metrics (Tutti i parametri).
3. Nella scheda Sfoglia, utilizzando il menu a discesa senza etichetta sulla sinistra, seleziona la propria regione "di origine", ovvero dove sono stati creati i canali. Per ulteriori informazioni sulle Regioni, consultare Soluzione globale, controllo regionale. Per un elenco delle Regioni supportate, consultare la pagina di Amazon IVS nei Riferimenti generali di AWS.
4. Nella parte inferiore della scheda Sfoglia, seleziona lo spazio dei nomi IVSRealTime.
5. Esegui una di queste operazioni:
   a. Nella barra di ricerca digitare l'ID della risorsa (parte dell'ARN, `arn:::ivs:stage/<resource id>`).

      Quindi seleziona IVSRealTime > Parametri fase.
   b. Se IVSRealTime viene visualizzato come servizio selezionabile in Spazi dei nomi AWS, selezionalo. Verrà elencato se utilizzi lo streaming in tempo reale di Amazon IVS e se invia i parametri ad Amazon CloudWatch. (Se IVSRealTime non è presente nell'elenco, allora significa che non si dispone di parametri Amazon IVS).

Selezione ora il raggruppamento di dimensioni desiderato. Le dimensioni disponibili sono elencate nei Parametri di CloudWatch qui sotto.

6. Seleziona i parametri da aggiungere al grafico. I parametri disponibili sono elencati nei Parametri di CloudWatch qui sotto.

È inoltre possibile accedere al grafico CloudWatch della sessione di streaming dalla pagina dei dettagli della sessione di streaming, selezionando il pulsante View in CloudWatch (Visualizza in CloudWatch).

Istruzioni per la CLI

È possibile accedere ai parametri anche utilizzando l'interfaccia a riga di comando (CLI) di AWS. Ciò richiede il download e la configurazione della CLI sul computer. Per maggiori dettagli, consultare la Guida per l'utente dell'interfaccia a riga di comando di AWS.

Quindi, per accedere ai parametri dello streaming in tempo reale di Amazon IVS utilizzando la CLI di AWS:

- Al prompt dei comandi, esegui:

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

Per ulteriori informazioni, consulta Utilizzo di parametri di Amazon CloudWatch nella Guida per l'utente di Amazon CloudWatch.

Parametri di CloudWatch: streaming in tempo reale IVS.

Amazon IVS fornisce i parametri riportati di seguito nello spazio nomi AWS/IVSRealTime.

Affinché i parametri di CloudWatch siano disponibili, è necessario utilizzare l'SD di trasmissione Web 1.5.2 o versione successiva.

La dimensione può avere i seguenti valori validi:

- La dimensione `Stage` è un ID di risorsa (parte dell'ARN, `arn:::stage/<resource id>`).
- La dimensione `Participant` è un `participantID`.

- `SimulcastLayer` è "elevato", "medio", "basso" o "no-rid" per un `MediaType` di "video" oppure "disabilitato" per un `MediaType` "audio." Questo valore può essere vuoto.
- La dimensione `MediaType` è "video" o "audio" (stringa).

Metrica	Dimensione	Description
`DownloadPacketLoss`	`Stage`	Ogni esempio rappresenta la percentuale di pacchetti persi da un determinato abbonato durante il download da un server IVS. Unità: percentuale Statistiche valide: media, massimo, minimo. Il numero medio, il numero più grande o il numero più piccolo (rispettivamente) di pacchetti persi nell'intervallo configurato
`DownloadPacketLoss`	`Stage,Participant`	Filtri `DownloadPacketLoss` per partecipante, per gli abbonati che sono anche editori. Gli esempi rappresentano la percentuale di pacchetti persi da un abbonato durante il download da un server IVS. I campioni vengono emessi solo quando il partecipante è anche un editore. Unità: percentuale Statistiche valide: media, massimo, minimo. Il numero medio, il numero più grande o il numero più piccolo (rispettivamente) di frame eliminati nell'intervallo configurato
`DroppedFrames`	`Stage`	Ogni esempio rappresenta la percentuale di frame che sono stati eliminati da un determinato abbonato. Unità: percentuale Statistiche valide: media, massimo, minimo. Il numero medio, il numero più grande o il numero più piccolo

Metrica	Dimensione	Description
		(rispettivamente) di frame eliminati nell'intervallo configurato
`DroppedFrames`	`Stage,Participant`	Filtri `DroppedFrames` per partecipante, per gli abbonati che sono anche editori. Gli esempi rappresentano la percentuale di fotogrammi interrotti tra il partecipante abbonato e tutti gli editori presenti sullo stage. I campioni vengono emessi solo quando il partecipante è anche un editore. Unità: percentuale Statistiche valide: media, massimo, minimo. Il numero medio, il numero più grande o il numero più piccolo (rispettivamente) di frame eliminati nell'intervallo configurato
`PublishBitrate`	`Stage`	I campioni emessi rappresentano la velocità totale con cui un determinato editore invia dati video e audio (sommati tra tutti i livelli di simulcast). Unità: bit al secondo Statistiche valide: media, massimo, minimo: il numero medio, il numero più grande o il numero più piccolo (rispettivamente) di bitrate nell'intervallo configurato

Metrica	Dimensione	Description
`PublishBitrate`	`Stage, Participant, SimulcastLayer, MediaType`	Filtri `PublishBitrate` per partecipante, livello di simulcast e tipo di supporto. L'ID del layer simulcast è impostato dall'SDK di trasmissione. Quando il simulcast è disabilitato, questo ID di livello verrà impostato su "disabilitato". Il tipo di supporto è video o audio. Unità: bit al secondo Statistiche valide: media, massimo, minimo: il numero medio, il numero più grande o il numero più piccolo (rispettivamente) di bitrate nell'intervallo configurato
`Publishers`	`Stage`	Numero di partecipanti che pubblicano sulla fase. Unità: numero Statistiche valide: Media, minimo, massimo
`PublishResolution`	`Stage, Participant, SimulcastLayer, MediaType`	Numero di pixel sul lato inferiore della larghezza o dell'altezza della cornice. Ad esempio, per un frame orizzontale di dimensioni 1920x1080, PublishResolution è 1080. Per un frame verticale di dimensioni 720x1280, la PublishResolution è 720. Unità: numero Statistiche valide: Media, minimo, massimo
`SubscribeBitrate`	`Stage`	I campioni emessi rappresentano la velocità totale alla quale un determinato abbonato riceve dati audio e video. Unità: bit al secondo Statistiche valide: media, massimo, minimo: il numero medio, il numero più grande o il numero più piccolo (rispettivamente) di bitrate nell'intervallo configurato

Metrica	Dimensione	Description
`SubscribeBitrate`	`Stage,Participant,MediaType`	Filtri `SubscribeBitrate` per partecipante, per gli abbonati che sono anche editori. Gli esempi rappresentano il bitrate per il quale un determinato abbonato riceve il dato `MediaType`. Gli esempi vengono emessi solo durante la pubblicazione da parte del partecipante abbonato. Unità: bit al secondo Statistiche valide: media, massimo, minimo: il numero medio, il numero più grande o il numero più piccolo (rispettivamente) di bitrate nell'intervallo configurato
`Subscribers`	`Stage`	Numero di partecipanti abbonati alla fase. Tieni presente che i partecipanti che pubblicano e si abbonano attivamente vengono conteggiati sia come editori che come abbonati. Unità: numero Statistiche valide: media, minimo, massimo

SDK di trasmissione IVS (streaming in tempo reale)

L'SDK di trasmissione dello streaming in tempo reale di Amazon Interactive Video Services (IVS) è rivolto agli sviluppatori che creano applicazioni con Amazon IVS. Questo SDK è progettato per trarre vantaggio dall'architettura di Amazon IVS e, così come Amazon IVS, vedrà l'introduzione di miglioramenti continui e nuove funzionalità. Essendo un SDK di trasmissione nativo, è progettato per ridurre al minimo l'impatto sulle prestazioni dell'applicazione e dei dispositivi utilizzati dagli utenti per accedere all'applicazione.

Tieni presente che l'SDK di trasmissione viene utilizzato sia per l'invio che per la ricezione di video, ovvero viene utilizzato lo stesso SDK sia per gli host che per gli spettatori. Non è necessario un SDK per lettori separati.

L'applicazione può avvalersi delle funzionalità principali dell'SDK di trasmissione Amazon IVS:

- Streaming di alta qualità - L'SDK di trasmissione supporta lo streaming di alta qualità. Cattura video dalla tua fotocamera e codificali fino a 720p.
- Regolazioni automatiche del bitrate - Gli utenti di smartphone sono mobili, quindi le loro condizioni di rete possono cambiare nel corso della trasmissione. L'SDK di trasmissione di Amazon IVS regola automaticamente il bitrate video per adattarsi alle mutevoli condizioni di rete.
- Supporto per l'orientamento verticale e orizzontale - Indipendentemente dal modo in cui gli utenti tengono in mano i dispositivi, l'immagine viene visualizzata e ridimensionata correttamente. L'SDK di trasmissione supporta ogni dimensione del riquadro, sia in verticale che in orizzontale. Gestisce automaticamente le sue proporzioni quando gli utenti ruotano il dispositivo e cambiano l'orientamento configurato.
- Streaming sicuro - Le trasmissioni dell'utente sono crittografate tramite TLS, in modo che possano mantenere protetti i propri flussi.
- Dispositivi audio esterni - L'SDK di trasmissione Amazon IVS supporta collegamenti audio con cavo, USB e microfoni esterni Bluetooth SCO.

Requisiti della piattaforma

Piattaforme native

Piattaforma	Versioni supportate
Android	9.0 e versioni successive: i clienti possono creare con la versione 5.0 ma non saranno in grado di utilizzare la funzionalità di streaming in tempo reale.
iOS	14 e versioni successive

IVS supporta un minimo di 4 versioni principali di iOS e 6 versioni principali di Android. Il nostro supporto per le versioni correnti potrebbe estendersi oltre questi minimi. I clienti verranno avvisati tramite note di rilascio dell'SDK con almeno 3 mesi di anticipo se una versione principale non è più supportata.

Browser desktop

Browser	Piattaforme supportate	Versioni supportate
Chrome	Windows, macOS	Due versioni principali (versione corrente e precedente più recente)
Firefox	Windows, macOS	Due versioni principali (versione corrente e precedente più recente)
Edge	Windows 8.1 e versioni successive	Due versioni principali (versione corrente e precedente più recente) Esclude Edge Legacy
Safari	macOS	Due versioni principali (versione corrente e precedente più recente)

Browser per dispositivi mobili (iOS e Android)

Browser	Piattaforme supportate	Versioni supportate
Chrome	iOS, Android	Due versioni principali (versione corrente e precedente più recente)
Firefox	Android	Due versioni principali (versione corrente e precedente più recente)
Safari	iOS	Due versioni principali (versione corrente e precedente più recente)

Limiti noti

- Su tutti i dispositivi mobili, sconsigliamo di pubblicare/abbonarsi con quattro o più partecipanti contemporaneamente, a causa di problemi relativi agli artefatti video e alle schermate nere. Se hai bisogno di più partecipanti, configura la pubblicazione e la sottoscrizione solo audio
- Non è consigliabile comporre una fase e trasmetterla su un canale su Android Mobile Web, a causa di considerazioni relative alle prestazioni e ai potenziali arresti anomali. Se è richiesta la funzionalità di trasmissione, integra l'SDK di trasmissione per lo streaming in tempo reale IVS per Android.

Viste Web

L'SDK di trasmissione Web non fornisce supporto per visualizzazioni Web o ambienti simili al Web (TV, console e così via). Per le implementazioni su dispositivi mobili, consulta la Guida all'SDK di trasmissione in streaming in tempo reale per Android e iOS.

Richiesta di accesso al dispositivo

L'SDK di trasmissione richiede l'accesso alle fotocamere e ai microfoni del dispositivo, sia quelli integrati nel dispositivo che quelli collegati tramite Bluetooth, USB o ingresso audio.

Supporto

L'SDK di trasmissione viene continuamente migliorato. Consultare le Note di rilascio di Amazon IVSper le versioni disponibili e i problemi risolti. Se necessario, prima di contattare il supporto, aggiornare la versione dell'SDK di trasmissione e verificare se il problema è stato risolto.

Controllo delle versioni

Gli SDK di trasmissione di Amazon IVS utilizzano il controllo semantico delle versioni.

Per questa discussione, supponiamo che:

- La versione più recente sia la 4.1.3.
- L'ultima versione della versione principale precedente sia 3.2.4.
- La versione più recente della versione 1.x sia la 1.5.6.

Le nuove funzionalità compatibili con le versioni precedenti vengono aggiunte come versioni secondarie dell'ultima versione. In questo caso, il set successivo di nuove funzionalità verrà aggiunto come versione 4.2.0.

Le correzioni di bug minori compatibili con le versioni precedenti vengono aggiunte come versioni di patch dell'ultima versione. Nel nostro caso, il set di correzioni minori di bug successivo sarà aggiunto come versione 4.1.4.

Le correzioni di bug principali compatibili con le versioni precedenti sono gestite in modo diverso, ovvero vengono aggiunte alle diverse versioni:

- Rilascio della patch dell'ultima versione. Nel nostro caso, questa è la versione 4.1.4.
- Rilascio della patch della versione secondaria precedente. Nel nostro caso, questa è la versione 3.2.5.
- Rilascio di patch dell'ultima versione 1.x. Nel nostro caso, questa è la versione 1.5.7.

Le correzioni di bug principali sono definite dal team di prodotti Amazon IVS. Esempi tipici sono gli aggiornamenti critici della sicurezza e alcune altre correzioni necessarie per i clienti.

Nota: negli esempi precedenti, le versioni rilasciate vengono incrementate senza saltare alcun numero (ad esempio, da 4.1.3 a 4.1.4). In realtà, uno o più numeri di patch possono rimanere interni e non essere rilasciati, quindi la versione rilasciata potrebbe aumentare da 4.1.3 a, ad esempio, 4.1.6.

SDK di trasmissione IVS: Guida per il Web (streaming in tempo reale)

L'SDK di trasmissione Web in tempo reale di IVS offre agli sviluppatori gli strumenti per creare esperienze interattive e in tempo reale sul Web. Questo SDK è rivolto agli sviluppatori che creano applicazioni Web con Amazon IVS.

L'SDK per la trasmissione Web consente ai partecipanti di inviare e ricevere video. L'SDK supporta le seguenti operazioni:

- Partecipa a uno stage
- Pubblica contenuti multimediali per gli altri partecipanti allo stage
- Iscriviti ai contenuti multimediali degli altri partecipanti allo stage
- Gestisci e monitora video e audio pubblicati sullo stage
- Ottieni statistiche WebRTC per ogni connessione peer
- Tutte le operazioni dell'SDK di trasmissione Web in streaming a bassa latenza di IVS

Ultima versione di Web broadcast SDK: 1.8.0 (Note di rilascio)

Documentazione di riferimento: per informazioni sui metodi più importanti disponibili in Amazon IVS Web Broadcast SDK, consulta https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference. Assicurati che sia selezionata la versione più recente dell'SDK.

Codice di esempio: gli esempi seguenti sono un buon punto di partenza per iniziare a utilizzare rapidamente l'SDK:

- HTML e JavaScript
- React

Requisiti della piattaforma: consulta SDK di trasmissione Amazon IVS per un elenco delle piattaforme supportate

Nozioni di base

Importazioni

Gli elementi costitutivi per il tempo reale si trovano in uno spazio dei nomi diverso da quello dei moduli di trasmissione principali.

Utilizzo di un tag di script

Utilizzando le stesse importazioni di script, le classi e le enumerazioni definite negli esempi seguenti sono individuabili sull'oggetto globale `IVSBroadcastClient`:

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

Utilizzo di npm

Le classi, le enumerazioni e i tipi possono essere importati anche dal modulo del pacchetto:

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

Richiedere autorizzazioni

L'app deve richiedere l'autorizzazione per accedere alla fotocamera e al microfono dell'utente e tale autorizzazione deve utilizzare HTTPS. (Questo non riguarda solo Amazon IVS, ma qualsiasi sito Web che abbia bisogno di accedere alle fotocamere e ai microfoni.)

Ecco un esempio di funzione che mostra come richiedere e ottenere le autorizzazioni per dispositivi audio e video:

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

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

Per ulteriori informazioni, consulta l'API Permissions eMediaDevices. getUserMedia().

Elenco dei dispositivi disponibili

Per vedere quali dispositivi sono disponibili per l'acquisizione, interrogate il MediaDevicesmetodo.enumerateDevices () del browser:

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

Recupera un file da un dispositivo MediaStream

Dopo aver acquisito l'elenco dei dispositivi disponibili, puoi recuperare un flusso da qualsiasi numero di dispositivi. Ad esempio, puoi utilizzare il metodo `getUserMedia()` per recuperare un flusso da una videocamera.

Se desideri specificare da quale dispositivo catturare lo streaming, puoi impostare esplicitamente il `deviceId` nella sezione `audio` o `video` dei vincoli del supporto. In alternativa, puoi omettere `deviceId` e fare in modo che gli utenti selezionino i propri dispositivi dal prompt del browser.

È inoltre possibile specificare una risoluzione ideale della fotocamera utilizzando i vincoli `width` e `height`. (Ulteriori informazioni su questi vincoli sono disponibili qui.) L'SDK applica automaticamente i limiti di larghezza e altezza che corrispondono alla risoluzione massima di trasmissione; tuttavia, è una buona idea applicarli anche tu stesso in modo da essere certi che le proporzioni dell'aspetto della sorgente non vengano modificate dopo aver aggiunto la sorgente all'SDK.

Per lo streaming in tempo reale, assicurati che i contenuti multimediali siano limitati alla risoluzione di 720p. In particolare, i valori dei `getDisplayMedia` vincoli di larghezza `getUserMedia` e altezza non devono superare 921600 (1280*720) se moltiplicati tra loro.

```
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 },
});
```

Pubblicazione e sottoscrizione

Concetti

La funzionalità in tempo reale si basa su tre concetti fondamentali: fase, strategia ed eventi. L'obiettivo di progettazione è ridurre al minimo la quantità di logica lato client necessaria per creare un prodotto funzionante.

Stage

La classe `Stage` è il principale punto di interazione tra l'applicazione host e l'SDK. Rappresenta lo stage stesso e serve per entrare e uscire dallo stage. La creazione e la partecipazione a uno stage richiedono una stringa di token valida e non scaduta dal piano di controllo (control-plane) (rappresentata come `token`). Entrare e uscire da uno stage è semplice:

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

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

stage.leave();
```

Strategia

L'interfaccia `StageStrategy` consente all'applicazione host di comunicare lo stato desiderato dello stage all'SDK. È necessario implementare tre funzioni: `shouldSubscribeToParticipant`, `shouldPublishParticipant` e `stageStreamsToPublish`. Sono tutte analizzate di seguito.

Per utilizzare una strategia definita, passala al costruttore `Stage`. Quello che segue è un esempio completo di applicazione che utilizza una strategia per pubblicare la webcam di un partecipante sullo stage ed eseguire la sottoscrizione a tutti i partecipanti. Lo scopo di ciascuna funzione strategica necessaria è spiegato in modo dettagliato nelle sezioni seguenti.

```
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();
```

Sottoscrizione ai partecipanti

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

Quando un partecipante remoto partecipa allo stage, l'SDK interroga l'applicazione host sullo stato della sottoscrizione desiderato per quel partecipante. Le opzioni sono `NONE`, `AUDIO_ONLY` e `AUDIO_VIDEO`. Quando si restituisce un valore per questa funzione, l'applicazione host non deve preoccuparsi dello stato di pubblicazione, dello stato della sottoscrizione corrente o dello stato della connessione allo stage. Se viene restituito `AUDIO_VIDEO`, l'SDK attende che il partecipante remoto effettui la pubblicazione prima della sottoscrizione e aggiorna l'applicazione host emettendo eventi durante tutto il processo.

Di seguito è riportata un'implementazione di esempio:

```
const strategy = {

   shouldSubscribeToParticipant: (participant) => {
```

```
      return SubscribeType.AUDIO_VIDEO;
   }

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

Questa è l'implementazione completa di questa funzione per un'applicazione host che vuole sempre che tutti i partecipanti si vedano tra loro, ad esempio un'applicazione di chat video.

Sono possibili anche implementazioni più avanzate. Utilizza la proprietà `userInfo` su `ParticipantInfo` per iscriverti selettivamente ai partecipanti in base agli attributi forniti dal server:

```
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
}
```

Questa può essere usata per creare uno stage in cui i moderatori possano monitorare tutti gli ospiti senza essere visti o ascoltati. L'applicazione host potrebbe utilizzare una logica aziendale aggiuntiva per consentire ai moderatori di vedersi ma rimanendo invisibili agli ospiti.

Pubblicazione

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

Una volta connesso allo stage, l'SDK interroga l'applicazione host per vedere se un dato partecipante deve eseguire una pubblicazione. Viene richiamata solo per i partecipanti locali che hanno il permesso di pubblicare in base al token fornito.

Di seguito è riportata un'implementazione di esempio:

```
const strategy = {

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

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

Si tratta di un'applicazione di chat video standard in cui gli utenti vogliono sempre pubblicare. Possono disattivare e riattivare l'audio e il video per essere nascosti o visti/ascoltati immediatamente. Possono anche usare il comando di pubblicazione/annullamento della pubblicazione, ma è molto più lento. È preferibile disattivare/riattivare l'audio nei casi d'uso in cui è consigliabile modificare spesso la visibilità.

Scelta dei flussi da pubblicare

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

Durante la pubblicazione, serve a determinare quali flussi audio e video devono essere pubblicati. Questo argomento verrà trattato dettagliatamente più avanti in Pubblicazione di un flusso multimediale.

Aggiornamento della strategia

La strategia è pensata per essere dinamica: è possibile modificare i valori restituiti da una qualsiasi delle funzioni precedenti in qualsiasi momento. Ad esempio, se l'applicazione host non desidera pubblicare finché l'utente finale non tocca un pulsante, è possibile restituire una variabile da `shouldPublishParticipant` (del tipo `hasUserTappedPublishButton`). Quando quella variabile cambia in base a un'interazione da parte dell'utente finale, chiama `stage.refreshStrategy()` per segnalare all'SDK che dovrebbe eseguire una query sulla strategia per i valori più recenti, applicando solo quanto modificato. Se l'SDK rileva che il valore `shouldPublishParticipant` è cambiato, avvia il processo di pubblicazione. Se le query dell'SDK e tutte le funzioni restituiscono lo stesso valore di prima, la chiamata `refreshStrategy` non modifica lo stage.

Se il valore restituito di `shouldSubscribeToParticipant` cambia da `AUDIO_VIDEO` a `AUDIO_ONLY`, il flusso video viene rimosso per tutti i partecipanti con valori restituiti modificati, se in precedenza esisteva un flusso video.

In genere, lo stage utilizza la strategia per applicare in modo più efficiente la differenza tra le strategie precedenti e quelle attuali, senza che l'applicazione host debba preoccuparsi di tutto lo stato necessario per gestirla correttamente. Per questo motivo, considera la chiamata a `stage.refreshStrategy()` come un'operazione a basso costo, perché non viene eseguita a meno che la strategia non cambi.

Eventi

Un'istanza `Stage` è un emettitore di eventi. Usando `stage.on()`, lo stato dello stage viene comunicato all'applicazione host. Gli aggiornamenti all'interfaccia utente dell'applicazione host in genere possono essere supportati interamente dagli eventi. Gli eventi sono i seguenti:

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

Per la maggior parte di questi metodi, viene fornito il `ParticipantInfo` corrispondente.

Non è previsto che le informazioni fornite dagli eventi influiscano sui valori restituiti della strategia. Ad esempio, non è previsto che il valore restituito di `shouldSubscribeToParticipant` cambi quando viene chiamato `STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`. Se l'applicazione host desidera effettuare la sottoscrizione a un particolare partecipante, deve restituire il tipo di abbonamento desiderato indipendentemente dallo stato di pubblicazione di quel partecipante. L'SDK è responsabile di garantire che lo stato desiderato della strategia venga applicato al momento giusto in base allo stato dello stage.

Pubblicazione di un flusso multimediale

I dispositivi locali come microfoni e fotocamere vengono recuperati utilizzando gli stessi passaggi descritti sopra in Recuperare un messaggio da un dispositivo. MediaStream Nell'esempio utilizziamo `MediaStream` per creare un elenco di oggetti `LocalStageStream` utilizzati per la pubblicazione dall'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
    }
}
```

Pubblicazione di una condivisione dello schermo

Spesso le applicazioni devono pubblicare una condivisione dello schermo in aggiunta alla webcam dell'utente. La pubblicazione di una condivisione dello schermo richiede la creazione di una `Stage` aggiuntiva con un proprio token univoco.

```
// 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();
```

Visualizzazione e rimozione dei partecipanti

Una volta completata la sottoscrizione, riceverai una serie di oggetti `StageStream` tramite l'evento `STAGE_PARTICIPANT_STREAMS_ADDED`. L'evento fornisce anche informazioni sui partecipanti per aiutarti a visualizzare i flussi multimediali:

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

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

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

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

Quando un partecipante interrompe la pubblicazione o annulla l'iscrizione a un flusso, la funzione `STAGE_PARTICIPANT_STREAMS_REMOVED` viene chiamata con i flussi che sono stati rimossi. Le applicazioni host devono utilizzarlo come segnale per rimuovere il flusso video del partecipante dal DOM.

`STAGE_PARTICIPANT_STREAMS_REMOVED` viene richiamato per tutti gli scenari in cui un flusso potrebbe essere rimosso, tra cui:

- Il partecipante remoto interrompe la pubblicazione.
- Un dispositivo locale annulla l'iscrizione o modifica l'abbonamento da `AUDIO_VIDEO` a `AUDIO_ONLY`.
- Il partecipante remoto lascia lo stage.
- Il partecipante locale lascia lo stage.

Poiché STAGE_PARTICIPANT_STREAMS_REMOVED viene richiamato per tutti gli scenari, non è richiesta alcuna logica aziendale personalizzata per la rimozione dei partecipanti dall'interfaccia utente durante le operazioni di abbandono remote o locali.

Disattivazione e riattivazione dell'audio dei flussi multimediali

Gli oggetti LocalStageStream hanno una funzione setMuted che controlla se l'audio del flusso è disattivato. Questa funzione può essere richiamata sul flusso prima o dopo la restituzione dalla funzione della strategia stageStreamsToPublish.

Importante: se una nuova istanza di oggetto LocalStageStream viene restituita da stageStreamsToPublish dopo una chiamata a refreshStrategy, lo stato di silenziamento del nuovo oggetto di flusso viene applicato allo stage. Fai attenzione quando crei nuove istanze LocalStageStream per assicurarti che lo stato di silenziamento previsto venga mantenuto.

Monitoraggio dello stato di silenziamento dei contenuti multimediali dei partecipanti remoti

Quando i partecipanti modificano lo stato di silenziamento del video o dell'audio, l'evento STAGE_STREAM_MUTE_CHANGED viene attivato con un elenco di flussi che sono stati modificati. Usa la proprietà isMuted su StageStream per aggiornare l'interfaccia utente di conseguenza:

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

Inoltre, puoi verificare se l'audio o il video sono disattivati StageParticipantInfoper verificare se l'audio o il video sono disattivati:

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

Ottenimento delle statistiche WebRTC

Per ottenere le statistiche WebRTC più recenti per un flusso di pubblicazione o un flusso di iscrizione, usa `getStats` su `StageStream`. Si tratta di un metodo asincrono con il quale è possibile recuperare le statistiche tramite attesa o concatenando una promessa. Il risultato è un `RTCStatsReport`, ovvero un dizionario contenente tutte le statistiche standard.

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

Ottimizzazione dei contenuti multimediali

Si consiglia di limitare le chiamate `getUserMedia` e `getDisplayMedia` secondo i seguenti vincoli per ottenere prestazioni ottimali:

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

È possibile limitare ulteriormente i contenuti multimediali tramite opzioni aggiuntive passate al costruttore `LocalStageStream`:

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

Nel codice qui sopra:

- `minBitrate` imposta un bitrate minimo che dovrebbe essere utilizzato dal browser. Tuttavia, un flusso video a bassa complessità può spingere il codificatore a scendere al di sotto di questo bitrate.
- `maxBitrate` imposta un bitrate massimo che il browser non dovrebbe superare per questo flusso.
- `maxFramerate` imposta una frequenza di rate massima che il browser non dovrebbe superare per questo flusso.
- L'opzione `simulcast` può essere utilizzata solo sui browser basati su Chromium. Consente l'invio di tre livelli di rendering del flusso.
  - Ciò consente al server di scegliere quale rendering inviare agli altri partecipanti in base alle loro limitazioni di rete.
  - Quando `simulcast` viene specificato insieme a un valore di `maxBitrate` e/o `maxFramerate`, si prevede che il livello di rendering più alto venga configurato tenendo conto di questi valori, a condizione che `maxBitrate` non scenda al di sotto del valore predefinito interno `maxBitrate` del secondo livello più alto dell'SDK di 900 kbps.
  - Se viene specificato un valore troppo basso di `maxBitrate` rispetto al valore predefinito del secondo livello più alto, `simulcast` sarà disabilitato.
  - `simulcast` non può essere attivato e disattivato senza ripubblicare il file multimediale tramite una sequenza in cui `shouldPublishParticipant` restituisce `false`, richiama `refreshStrategy`, `shouldPublishParticipant` restituisce `true` e richiama di nuovo `refreshStrategy`.

Ottieni gli attributi dei partecipanti

Se specifichi gli attributi nella richiesta dell'endpoint `CreateParticipantToken`, puoi visualizzarli nelle proprietà `StageParticipantInfo`:

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

Gestione dei problemi di rete

Quando si perde la connessione di rete del dispositivo locale, l'SDK tenta internamente di riconnettersi senza alcuna azione da parte dell'utente. In alcuni casi, l'SDK non funziona ed è necessaria un'azione da parte dell'utente.

In generale, lo stato dello stage può essere gestito tramite l'evento `STAGE_CONNECTION_STATE_CHANGED`:

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

In generale, il verificarsi di errori dopo l'accesso corretto a uno stage indica che l'SDK non risulta connesso e non è riuscito a ristabilire una connessione. Crea un nuovo oggetto `Stage` e prova ad accedere quando le condizioni della rete migliorano.

Trasmissione della fase a un canale IVS

Per trasmettere uno stage, creare una sessione `IVSBroadcastClient` separata e segui le consuete istruzioni per la trasmissione con l'SDK descritte sopra. L'elenco dei `StageStream` esposti tramite `STAGE_PARTICIPANT_STREAMS_ADDED` può essere utilizzato per recuperare i flussi multimediali dei partecipanti che possono essere applicati alla composizione del flusso di trasmissione, come segue:

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

Facoltativamente, puoi comporre una fase e trasmetterla a un canale IVS a bassa latenza in modo da raggiungere un pubblico più vasto. Consulta Abilitazione di più host su un flusso Amazon IVS nella Guida per l'utente dello streaming a bassa latenza di IVS.

Problemi noti e soluzioni alternative

- Quando si chiudono le schede o si esce dal browser senza chiamare `stage.leave()`, gli utenti possono comunque apparire nella sessione con un frame bloccato o una schermata nera per un massimo di 10 secondi.

  Soluzione alternativa: nessuna.

- Le sessioni di Safari vengono visualizzate in modo intermittente con una schermata nera agli utenti che si iscrivono dopo l'inizio di una sessione.

  Soluzione alternativa: aggiorna il browser e ricollega la sessione.

- Quando si passa da una rete all'altra, il ripristino di Safari non avviene correttamente.

  Soluzione alternativa: aggiorna il browser e ricollega la sessione.

- La console per sviluppatori ripete un errore `Error: UnintentionalError at StageSocket.onClose`.

  Soluzione alternativa: è possibile creare un solo stage per token di partecipazione. Questo errore si verifica quando viene creata più di un'istanza `Stage`con lo stesso token di partecipazione, indipendentemente dal fatto che l'istanza si trovi su uno o più dispositivi.

- Potresti avere problemi a mantenere uno `StageParticipantPublishState.PUBLISHED` stato e potresti ricevere `StageParticipantPublishState.ATTEMPTING_PUBLISH` stati ripetuti durante l'ascolto dell'`StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`evento.

Soluzione alternativa: limita la risoluzione video a 720p quando si richiama o. `getUserMedia` `getDisplayMedia` In particolare, i valori dei `getDisplayMedia` vincoli di larghezza `getUserMedia` e altezza non devono superare 921600 (1280*720) se moltiplicati tra loro.

Limiti di Safari

- Per negare una richiesta di autorizzazione è necessario reimpostare l'autorizzazione nelle impostazioni del sito Web di Safari a livello di sistema operativo.
- Safari non rileva nativamente tutti i dispositivi con la stessa efficacia di Firefox o Chrome. Ad esempio, OBS Virtual Camera non viene rilevata.

Limitazioni di Firefox

- Per consentire a Firefox di condividere lo schermo devono essere abilitate le autorizzazioni di sistema. Dopo averli abilitati, l'utente deve riavviare Firefox per farlo funzionare correttamente; in caso contrario, se le autorizzazioni vengono percepite come bloccate, il browser genererà un'eccezione. NotFoundError
- Manca il metodo `getCapabilities`. Ciò significa che gli utenti non possono ottenere la risoluzione o le proporzioni della traccia multimediale. Consulta questo thread di bugzilla.
- Mancano diverse proprietà `AudioContext`, ad esempio latenza e numero di canali. Ciò potrebbe rappresentare un problema per gli utenti esperti che desiderano manipolare le tracce audio.

- I feed della fotocamera da `getUserMedia` su MacOS sono limitati a un rapporto di aspetto 4:3. Consulta il thread 1 di bugzilla e il thread 2 di bugzilla.

- L'acquisizione audio non è supportata con `getDisplayMedia`. Consulta questo thread di bugzilla.
- La frequenza di fotogrammi nell'acquisizione dello schermo non è ottimale (circa 15 fps?). Consulta questo thread di bugzilla.

Limitazioni Web per dispositivi mobili

- getDisplayMedialla condivisione dello schermo non è supportata sui dispositivi mobili.

  Soluzione alternativa: nessuna.

- Il partecipante impiega 15-30 secondi per uscire quando chiude un browser senza chiamare `leave()`.

  Soluzione alternativa: aggiungi un'interfaccia utente che incoraggi gli utenti a disconnettersi correttamente.
- L'app in backgrounding interrompe la pubblicazione del video.

  Soluzione alternativa: visualizza uno stato dell'interfaccia utente quando l'editore è in pausa.
- La frequenza dei fotogrammi video diminuisce per circa 5 secondi dopo aver riattivato l'audio di una fotocamera su dispositivi Android.

  Soluzione alternativa: nessuna.
- Il feed video viene allungato in fase di rotazione per iOS 16.0.

  Soluzione alternativa: visualizza un'interfaccia utente che descrive questo problema noto del sistema operativo.
- La commutazione del dispositivo di ingresso audio commuta automaticamente il dispositivo di uscita audio.

  Soluzione alternativa: nessuna.
- Lo sfondo del browser fa sì che il flusso di pubblicazione diventi nero e produca solo audio.

  Soluzione alternativa: nessuna. Ciò è dovuto a motivi di sicurezza.

Gestione errori

Questa sezione fornisce una panoramica delle condizioni di errore, del modo in cui l'SDK di trasmissione Web le segnala all'applicazione e di cosa dovrebbe fare un'applicazione quando si verificano tali errori. Esistono quattro categorie di errori:

```
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
  }
});
```

Errori di creazione di istanze dello stage

La creazione di istanze dello stage non convalida in remoto i token, ma verifica alcuni problemi di base dei token che possono essere convalidati sul lato client. Di conseguenza, l'SDK potrebbe generare un errore.

Token partecipante non conforme

Ciò si verifica quando il token dello stage non è conforme. Quando si crea un'istanza di uno stage, l'SDK genera un errore con questo messaggio: "Errore durante l'analisi del token dello stage".

Azione: crea un token valido e riprova a creare un'istanza.

Errori di accesso allo stage

Questi sono gli errori che possono verificarsi quando si tenta inizialmente di accedere a uno stage.

Lo stage è stato eliminato

Ciò si verifica quando si entra in uno stage (associato a un token) che è stato eliminato. Il metodo `join` SDK genera un errore con questo messaggio: "InitialConnectTimedOut dopo 10 secondi».

Azione: crea un token valido con un nuovo stage e riprova a partecipare.

Token partecipante scaduto

Ciò si verifica quando il token è scaduto. Il metodo SDK `join` genera un errore con questo messaggio: "Il token è scaduto e non è più valido".

Azione: crea un nuovo token e riprova a partecipare.

Token partecipante non valido o revocato

Ciò si verifica quando il token non è valido o è stato revocato/disconnesso. Il metodo `join` SDK genera un errore con questo messaggio: "InitialConnectTimedOut dopo 10 secondi».

Azione: crea un nuovo token e riprova a partecipare.

Token disconnesso

Ciò si verifica quando il token dello stage è conforme ma viene rifiutato dal server degli stage. Il metodo `join` SDK genera un errore con questo messaggio: "InitialConnectTimedOut dopo 10 secondi».

Azione: crea un token valido e riprova a partecipare.

Errori di rete per l'accesso iniziale

Ciò si verifica quando l'SDK non riesce a contattare il server degli stage per stabilire una connessione. Il metodo `join` SDK genera un errore con questo messaggio: "InitialConnectTimedOut dopo 10 secondi».

Azione: attendi il ripristino della connettività del dispositivo e riprova a connetterti.

Errori di rete quando è già stato effettuato l'accesso

Se la connessione di rete del dispositivo si interrompe, l'SDK potrebbe perdere la connessione ai server dello stage. È possibile che vengano visualizzati errori nella console perché l'SDK non è più in grado di raggiungere i servizi di backend. I POST su https://broadcast.stats.live-video.net falliranno.

Se stai pubblicando e/o ti stai iscrivendo, vedrai errori nella console relativi ai tentativi di pubblicazione/sottoscrizione.

Internamente, l'SDK proverà a riconnettersi con una strategia di backoff esponenziale.

Azione: attendi il ripristino della connettività del dispositivo. In caso di pubblicazione o sottoscrizione, aggiorna la strategia per garantire la ripubblicazione dei tuoi stream multimediali.

Errori di pubblicazione e sottoscrizione

Errore di pubblicazione: stati di pubblicazione

L'SDK segnala `ERROR` quando una pubblicazione non va a buon fine. Ciò può verificarsi a causa delle condizioni della rete o se uno stage ha raggiunto la capacità massima di editori.

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

Azione: aggiorna la strategia per tentare di ripubblicare i tuoi flussi multimediali.

Errori di sottoscrizione

L'SDK segnala `ERROR` quando una sottoscrizione fallisce. Ciò può verificarsi a causa delle condizioni della rete o se uno stage ha raggiunto la capacità massima di abbonati.

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

Azione: aggiorna la strategia per provare una nuova sottoscrizione.

SDK di trasmissione IVS: Guida per Android (streaming in tempo reale)

L'SDK di trasmissione per lo streaming in tempo reale IVS per Android consente ai partecipanti di inviare e ricevere video su Android.

Il pacchetto `com.amazonaws.ivs.broadcast` implementa l'interfaccia descritta in questo documento. L'SDK supporta le seguenti operazioni:

- Partecipa a uno stage

- Pubblica contenuti multimediali per gli altri partecipanti allo stage
- Iscriviti ai contenuti multimediali degli altri partecipanti allo stage
- Gestisci e monitora video e audio pubblicati sullo stage
- Ottieni statistiche WebRTC per ogni connessione peer
- Tutte le operazioni dell'SDK di trasmissione per lo streaming a bassa latenza di IVS per Android

Ultima versione di Android broadcast SDK: 1.14.1 (Note di rilascio)

Documentazione di riferimento: per informazioni sui metodi più importanti disponibili nell'SDK di trasmissione Android di Amazon IVS, consulta la documentazione di riferimento all'indirizzo https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/android/.

Codice di esempio: consulta l'archivio di esempio per Android su: https://github.com/aws-samples/ - sample. GitHub amazon-ivs-broadcast-android

Requisiti della piattaforma: Android 9.0 e versioni successive.

Nozioni di base

Installare la libreria

Per aggiungere la libreria di trasmissione di Amazon IVS per Android al proprio ambiente di sviluppo Android, aggiungere la libreria al file `build.gradle` come mostrato di seguito (per l'ultima versione dell'SDK di trasmissione di Amazon IVS):

```
repositories {
    mavenCentral()
}

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

Aggiungi la seguente autorizzazione al tuo manifesto per consentire all'SDK di abilitare e disabilitare il vivavoce:

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

In alternativa, per installare manualmente l'SDK, scaricare la versione più recente da questo percorso:

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

Assicurati di scaricare `aar` con `-stages` aggiunto.

Richiedere autorizzazioni

L'app deve richiedere l'autorizzazione per accedere alla fotocamera e al microfono dell'utente. (Questo non riguarda solo Amazon IVS, ma qualsiasi applicazione che abbia bisogno di accedere alle fotocamere e ai microfoni.)

Qui, controlliamo se l'utente ha già concesso le autorizzazioni e, in caso contrario, le chiediamo:

```
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;
    }
}
```

Qui, otteniamo la risposta dell'utente:

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

Pubblicazione e sottoscrizione

Concetti

La funzionalità in tempo reale si basa su tre concetti fondamentali: fase, strategia e renderer. L'obiettivo di progettazione è ridurre al minimo la quantità di logica lato client necessaria per creare un prodotto funzionante.

Stage

La classe `Stage` è il principale punto di interazione tra l'applicazione host e l'SDK. Rappresenta lo stage stesso e serve per entrare e uscire dallo stage. La creazione e la partecipazione a uno stage richiedono una stringa di token valida e non scaduta dal piano di controllo (control-plane) (rappresentata come `token`). Entrare e uscire da uno stage è semplice.

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

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

stage.leave();
```

La classe `Stage` è anche il luogo in cui può essere collegato lo `StageRenderer`:

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

Strategia

L'interfaccia `Stage.Strategy` consente all'applicazione host di comunicare lo stato desiderato dello stage all'SDK. È necessario implementare tre funzioni: `shouldSubscribeToParticipant`, `shouldPublishFromParticipant` e `stageStreamsToPublishForParticipant`. Sono tutte analizzate di seguito.

Sottoscrizione ai partecipanti

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

Quando un partecipante remoto partecipa allo stage, l'SDK interroga l'applicazione host sullo stato della sottoscrizione desiderato per quel partecipante. Le opzioni sono `NONE`, `AUDIO_ONLY` e `AUDIO_VIDEO`. Quando si restituisce un valore per questa funzione, l'applicazione host non deve preoccuparsi dello stato di pubblicazione, dello stato della sottoscrizione corrente o dello stato della connessione allo stage. Se viene restituito `AUDIO_VIDEO`, l'SDK attende che il partecipante remoto effettui la pubblicazione prima della sottoscrizione e aggiorna l'applicazione host tramite il renderer durante tutto il processo.

Di seguito è riportata un'implementazione di esempio:

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

Questa è l'implementazione completa di questa funzione per un'applicazione host che vuole sempre che tutti i partecipanti si vedano tra loro, ad esempio un'applicazione di chat video.

Sono possibili anche implementazioni più avanzate. Utilizza la proprietà `userInfo` su `ParticipantInfo` per iscriverti selettivamente ai partecipanti in base agli attributi forniti dal server:

```
@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;
 }
}
```

Questa può essere usata per creare uno stage in cui i moderatori possano monitorare tutti gli ospiti senza essere visti o ascoltati. L'applicazione host potrebbe utilizzare una logica aziendale aggiuntiva per consentire ai moderati di vedersi, ma rimanendo invisibili agli ospiti.

Pubblicazione

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

Una volta connesso allo stage, l'SDK interroga l'applicazione host per vedere se un dato partecipante deve eseguire una pubblicazione. Viene richiamata solo per i partecipanti locali che hanno il permesso di pubblicare in base al token fornito.

Di seguito è riportata un'implementazione di esempio:

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

Si tratta di un'applicazione di chat video standard in cui gli utenti vogliono sempre pubblicare. Possono disattivare e riattivare l'audio e il video per essere nascosti o visti/ascoltati immediatamente. Possono anche usare il comando di pubblicazione/annullamento della pubblicazione, ma è molto più lento. È preferibile disattivare/riattivare l'audio nei casi d'uso in cui è consigliabile modificare spesso la visibilità.

Scelta dei flussi da pubblicare

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

Durante la pubblicazione, serve a determinare quali flussi audio e video devono essere pubblicati. Questo argomento verrà trattato dettagliatamente più avanti in Pubblicazione di un flusso multimediale.

Aggiornamento della strategia

La strategia è pensata per essere dinamica: è possibile modificare i valori restituiti da una qualsiasi delle funzioni precedenti in qualsiasi momento. Ad esempio, se l'applicazione host non desidera pubblicare finché l'utente finale non tocca un pulsante, è possibile restituire una variabile da `shouldPublishFromParticipant` (del tipo `hasUserTappedPublishButton`). Quando quella variabile cambia in base a un'interazione da parte dell'utente finale, chiama `stage.refreshStrategy()` per segnalare all'SDK che dovrebbe eseguire una query sulla strategia per i valori più recenti, applicando solo quanto modificato. Se l'SDK rileva che il valore `shouldPublishFromParticipant` è cambiato, avvierà il processo di pubblicazione. Se le query dell'SDK e tutte le funzioni restituiscono lo stesso valore di prima, la chiamata `refreshStrategy` non apporterà alcuna modifica allo stage.

Se il valore restituito di `shouldSubscribeToParticipant` cambia da `AUDIO_VIDEO` a `AUDIO_ONLY`, il flusso video verrà rimosso per tutti i partecipanti con valori restituiti modificati, se in precedenza esisteva un flusso video.

In genere, lo stage utilizza la strategia per applicare in modo più efficiente la differenza tra le strategie precedenti e quelle attuali, senza che l'applicazione host debba preoccuparsi di tutto lo stato necessario per gestirla correttamente. Per questo motivo, considera la chiamata a `stage.refreshStrategy()` come un'operazione a basso costo, perché non viene eseguita a meno che la strategia non cambi.

Renderer

L'interfaccia `StageRenderer` comunica lo stato desiderato dello stage all'applicazione host. Gli aggiornamenti all'interfaccia utente dell'applicazione host in genere possono essere alimentati interamente dagli eventi forniti dal renderer. Il renderer fornisce le funzioni seguenti:

```
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);
```

Per la maggior parte di questi metodi, vengono forniti `Stage` e `ParticipantInfo` corrispondenti.

Non è previsto che le informazioni fornite dal renderer influiscano sui valori restituiti della strategia. Ad esempio, non è previsto che il valore restituito di `shouldSubscribeToParticipant` cambi quando viene chiamato `onParticipantPublishStateChanged`. Se l'applicazione host desidera effettuare la sottoscrizione a un particolare partecipante, deve restituire il tipo di abbonamento desiderato indipendentemente dallo stato di pubblicazione di quel partecipante. L'SDK è responsabile di garantire che lo stato desiderato della strategia venga applicato al momento giusto in base allo stato dello stage.

Lo `StageRenderer` può essere collegato alla classe dello stage:

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

Ricorda che solo i partecipanti alla pubblicazione attivano `onParticipantJoined` e ogni volta che un partecipante interrompe la pubblicazione o abbandona la sessione dello stage viene attivato `onParticipantLeft`.

Pubblicazione di un flusso multimediale

I dispositivi locali, come microfoni e fotocamere integrati, vengono rilevati tramite `DeviceDiscovery`. Ecco un esempio di selezione della fotocamera frontale e del microfono predefinito, che vengono poi restituiti come `LocalStageStreams` per la pubblicazione dall'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;
}
```

Visualizzazione e rimozione dei partecipanti

Una volta completata la sottoscrizione, riceverai una serie di oggetti `StageStream` tramite la funzione on`StreamsAdded` del renderer. Puoi recuperare l'anteprima da 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);
```

Puoi recuperare le statistiche del livello audio da un `AudioStageStream`:

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

Quando un partecipante interrompe la pubblicazione o annulla l'iscrizione, la funzione `onStreamsRemoved` viene chiamata con i flussi che sono stati rimossi. Le applicazioni host devono utilizzarlo come segnale per rimuovere il flusso video del partecipante dalla gerarchia delle visualizzazioni.

`onStreamsRemoved` viene richiamato per tutti gli scenari in cui un flusso potrebbe essere rimosso, tra cui:

- Il partecipante remoto interrompe la pubblicazione.
- Un dispositivo locale annulla l'iscrizione o modifica l'abbonamento da `AUDIO_VIDEO` a `AUDIO_ONLY`.
- Il partecipante remoto lascia lo stage.
- Il partecipante locale lascia lo stage.

Poiché `onStreamsRemoved` viene richiamato per tutti gli scenari, non è richiesta alcuna logica aziendale personalizzata per la rimozione dei partecipanti dall'interfaccia utente durante le operazioni di abbandono remote o locali.

Disattivazione e riattivazione dell'audio dei flussi multimediali

Gli oggetti `LocalStageStream` hanno una funzione `setMuted` che controlla se l'audio del flusso è disattivato. Questa funzione può essere richiamata sul flusso prima o dopo la restituzione dalla funzione della strategia `streamsToPublishForParticipant`.

Importante: se una nuova istanza di oggetto `LocalStageStream` viene restituita da `streamsToPublishForParticipant` dopo una chiamata a `refreshStrategy`, lo stato di silenziamento del nuovo oggetto di flusso viene applicato allo stage. Fai attenzione quando crei nuove istanze `LocalStageStream` per assicurarti che lo stato di silenziamento previsto venga mantenuto.

Monitoraggio dello stato di silenziamento dei contenuti multimediali dei partecipanti remoti

Quando un partecipante modifica lo stato di silenziamento del proprio flusso video o audio, la funzione `onStreamMutedChanged` del renderer viene richiamata con un elenco di flussi che sono stati modificati. Usa il metodo `getMuted` su `StageStream` per aggiornare l'interfaccia utente di conseguenza.

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

Ottenimento delle statistiche WebRTC

Per ottenere le statistiche WebRTC più recenti per un flusso di pubblicazione o un flusso di iscrizione, usa `requestRTCStats` su `StageStream`. Quando una raccolta è completata, riceverai statistiche tramite il `StageStream.Listener` che può essere impostato su `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());
  }
 }
}
```

Ottieni gli attributi dei partecipanti

Se specifichi gli attributi nella richiesta dell'endpoint `CreateParticipantToken`, puoi visualizzarli nelle proprietà `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());
 }
}
```

Continuazione della sessione in background

Quando l'app entra in background, potresti voler interrompere la pubblicazione o iscriverti solo all'audio degli altri partecipanti remoti. A tale scopo, aggiorna l'implementazione `Strategy` per interrompere la pubblicazione e iscriviti a `AUDIO_ONLY` (o `NONE`, se applicabile).

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

Abilitazione/disabilitazione della codifica a livelli con simulcast

Quando si pubblica un flusso multimediale, l'SDK trasmette flussi video di alta e bassa qualità, pertanto i partecipanti remoti possono sottoscrivere il flusso anche se hanno una larghezza di banda limitata per il downlink. La codifica a livelli con simulcast è attiva per impostazione predefinita. Puoi disabilitarla usando la classe `StageVideoConfiguration.Simulcast`:

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

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

Limitazioni alla configurazione video

L'SDK non supporta l'uso forzato della modalità verticale o della modalità orizzontale `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)`. Nell'orientamento verticale, la dimensione più piccola viene utilizzata come larghezza; nell'orientamento orizzontale, l'altezza. Ciò significa che le due chiamate seguenti a `setSize` avranno lo stesso effetto sulla configurazione video:

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

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

Gestione dei problemi di rete

Quando si perde la connessione di rete del dispositivo locale, l'SDK tenta internamente di riconnettersi senza alcuna azione da parte dell'utente. In alcuni casi, l'SDK non funziona ed è necessaria un'azione da parte dell'utente. Esistono due errori principali relativi alla perdita della connessione di rete:

- Codice di errore 1400, messaggio: "PeerConnection è perso a causa di un errore di rete sconosciuto»
- Codice di errore 1300, messaggio: "Retry attempts are exhausted" (Nuovi tentativi esauriti)

Se viene ricevuto il primo errore ma il secondo no, l'SDK è ancora connesso allo stage e tenterà di ristabilire automaticamente le connessioni. Come misura di sicurezza, puoi chiamare `refreshStrategy` senza modificare i valori restituiti dal metodo di strategia, per attivare un tentativo di riconnessione manuale.

Se viene ricevuto il secondo errore, i tentativi di riconnessione dell'SDK sono falliti e il dispositivo locale non è più connesso allo stage. In questo caso, prova a rientrare nello stage chiamando `join` dopo aver ristabilito la connessione di rete.

In generale, il verificarsi di errori dopo l'accesso corretto a uno stage indica che l'SDK non è riuscito a ristabilire una connessione. Crea un nuovo oggetto `Stage` e prova ad accedere quando le condizioni della rete migliorano.

Uso dei microfoni Bluetooth

Per pubblicare utilizzando dispositivi microfonici Bluetooth, è necessario avviare una connessione Bluetooth SCO:

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

Problemi noti e soluzioni alternative

- Quando un dispositivo Android entra in modalità sospensione e si riattiva, è possibile che l'anteprima sia bloccata.

  Soluzione alternativa: crea e usa una nuova `Stage`.
- Quando un partecipante accede con un token utilizzato da un altro partecipante, la prima connessione viene disconnessa senza un errore specifico.

  Soluzione alternativa: nessuna.
- Può verificarsi un problema raro per cui l'editore sta pubblicando, ma lo stato di pubblicazione che gli abbonati ricevono è `inactive`.

  Soluzione alternativa: prova a uscire e poi a partecipare alla sessione. Se il problema persiste, crea un nuovo token per l'editore.

- Durante una sessione di stage può verificarsi un raro problema di distorsione audio a intermittenza, in genere durante le chiamate di lunga durata.

  Soluzione alternativa: il partecipante con audio distorto può uscire e accedere nuovamente alla sessione oppure annullare la pubblicazione e ripubblicare l'audio per risolvere il problema.
- I microfoni esterni non sono supportati durante la pubblicazione su uno stage.

  Soluzione alternativa: non utilizzare un microfono esterno collegato tramite USB per la pubblicazione su uno stage.
- La pubblicazione su uno stage con condivisione dello schermo usando `createSystemCaptureSources` non è supportata.

  Soluzione alternativa: gestisci l'acquisizione del sistema manualmente, utilizzando sorgenti di input di immagini personalizzate e sorgenti di input audio personalizzate.
- Quando una `ImagePreviewView` viene rimossa da un elemento padre (ad esempio, `removeView()` viene chiamato dall'elemento padre), `ImagePreviewView` viene rilasciata immediatamente. `ImagePreviewView` non mostra alcun frame quando viene aggiunta a un'altra vista principale.

  Soluzione alternativa: richiedi un'altra anteprima utilizzando `getPreview`.
- Quando partecipi a uno stage con un Samsung Galaxy S22/+ con Android 12, potresti riscontrare un errore 1401 e il dispositivo locale non riesce ad accedere allo stage o accede ma senza audio.

  Soluzione alternativa: esegui l'aggiornamento ad Android 13.
- Quando accedi a uno stage con un Nokia X20 su Android 13, la fotocamera potrebbe non aprirsi e viene generata un'eccezione.

  Soluzione alternativa: nessuna.
- I dispositivi con chipset MediaTek Helio potrebbero non riprodurre correttamente i video dei partecipanti remoti.

  Soluzione alternativa: nessuna.
- Su alcuni dispositivi, il sistema operativo del dispositivo può scegliere un microfono diverso da quello selezionato tramite l'SDK. Questo perché l'SDK di trasmissione Amazon IVS non può controllare come viene definito il percorso audio `VOICE_COMMUNICATION`, poiché varia in base ai diversi produttori di dispositivi.

  Soluzione alternativa: nessuna.

- Alcuni codificatori video Android non possono essere configurati con dimensioni video inferiori a 176x176. La configurazione di una dimensione inferiore causa un errore e impedisce lo streaming.

  Soluzione alternativa: non configurare la dimensione del video in modo che sia inferiore a 176x176.

Gestione errori

Errori irreversibili e non irreversibili

L'oggetto errore ha un campo booleano "è irreversibile" di `BroadcastException`.

In generale, gli errori irreversibili sono legati alla connessione al server degli stage (ad es. non è possibile stabilire una connessione oppure la connessione viene interrotta e non può essere recuperata). L'applicazione dovrebbe ricreare lo stage e riaccedervi, possibilmente con un nuovo token o quando la connettività del dispositivo viene ripristinata.

Gli errori non irreversibili sono generalmente correlati allo stato di pubblicazione/sottoscrizione e vengono gestiti dall'SDK, che riprova l'operazione di pubblicazione/sottoscrizione.

Puoi controllare questa proprietà:

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

Errori di accesso

Token non conforme

Ciò accade quando il token dello stage non è conforme.

L'SDK genera un'eccezione Java da una chiamata a `stage.join`, con codice di errore = 1000 e fatal = true.

Azione: crea un token valido e riprova a partecipare.

Token scaduto

Ciò accade quando il token dello stage è scaduto.

L'SDK genera un'eccezione Java da una chiamata a `stage.join`, con codice di errore = 1001 e fatal = true.

Azione: crea un nuovo token e riprova a partecipare.

Token non valido o revocato

Ciò accade quando il token dello stage è conforme ma viene rifiutato dal server degli stage. Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK chiama `onConnectionStateChanged` con un'eccezione, con codice di errore = 1026 e fatal = true.

Azione: crea un token valido e riprova a partecipare.

Errori di rete per l'accesso iniziale

Ciò accade quando l'SDK non riesce a contattare il server degli stage per stabilire una connessione. Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK chiama `onConnectionStateChanged` con un'eccezione, con codice di errore = 1300 e fatal = true.

Azione: attendi il ripristino della connettività del dispositivo e riprova a connetterti.

Errori di rete quando è già stato effettuato l'accesso

Se la connessione di rete del dispositivo si interrompe, l'SDK potrebbe perdere la connessione ai server dello stage. Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK chiama `onConnectionStateChanged` con un'eccezione, con codice di errore = 1300 e fatal = true.

Azione: attendi il ripristino della connettività del dispositivo e riprova a connetterti.

Errori di pubblicazione/sottoscrizione

Initial

Esistono diversi tipi di errori:

- MultihostSessionOfferCreationFailPublish (1020)
- MultihostSessionOfferCreationFailSubscribe (1021)

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

Questi vengono segnalati in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK riprova l'operazione per un numero limitato di volte. Durante i nuovi tentativi, lo stato di pubblicazione/sottoscrizione è `ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE`. Se i nuovi tentativi hanno esito positivo, lo stato diventa `PUBLISHED/SUBSCRIBED`.

L'SDK chiama `onError` con il codice di errore pertinente e fatal = false.

Azione: non è necessaria alcuna azione, poiché l'SDK riprova automaticamente. Facoltativamente, l'applicazione può aggiornare la strategia per imporre ulteriori tentativi.

Già stabilita, poi fallita

Una pubblicazione o una sottoscrizione possono fallire una volta stabilite, molto probabilmente a causa di un errore di rete. Il codice di errore per una "connessione peer interrotta a causa di un errore di rete" è 1400.

Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK riprova l'operazione di pubblicazione/sottoscrizione. Durante i nuovi tentativi, lo stato di pubblicazione/sottoscrizione è `ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE`. Se i nuovi tentativi hanno esito positivo, lo stato diventa `PUBLISHED/SUBSCRIBED`.

L'SDK chiama `onError` con il codice di errore = 1400 e fatal = false.

Azione: non è necessaria alcuna azione, poiché l'SDK riprova automaticamente. Facoltativamente, l'applicazione può aggiornare la strategia per imporre ulteriori tentativi. In caso di perdita totale della connettività, è probabile che anche la connessione agli stage fallisca.

SDK di trasmissione IVS: Guida per iOS (streaming in tempo reale)

L'SDK di trasmissione per lo streaming in tempo reale IVS per iOS consente ai partecipanti di inviare e ricevere video su iOS.

Il modulo `AmazonIVSBroadcast` implementa l'interfaccia descritta in questo documento. Sono supportate le seguenti operazioni:

- Partecipa a uno stage
- Pubblica contenuti multimediali per gli altri partecipanti allo stage
- Iscriviti ai contenuti multimediali degli altri partecipanti allo stage
- Gestisci e monitora video e audio pubblicati sullo stage
- Ottieni statistiche WebRTC per ogni connessione peer
- Tutte le operazioni dell'SDK di trasmissione in streaming a bassa latenza IVS per iOS

Ultima versione di iOS broadcast SDK: 1.14.1 (Note di rilascio)

Documentazione di riferimento: per informazioni sui metodi più importanti disponibili nell'SDK di trasmissione per iOS di Amazon IVS, consulta la documentazione di riferimento all'indirizzo https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios/.

Codice di esempio: vedi l'archivio di esempio iOS su GitHub: https://github.com/aws-samples/amazon-ivs-broadcast-ios -sample.

Requisiti della piattaforma: iOS 14 o superiore.

Nozioni di base

Installare la libreria

Ti consigliamo di integrare l'SDK di trasmissione tramite. CocoaPods (in alternativa, esiste la possibilità di aggiungere il framework al proprio progetto manualmente).

Consigliato: integra il Broadcast SDK () CocoaPods

La funzionalità in tempo reale è pubblicata come sottospecie dell'SDK di trasmissione in streaming a bassa latenza per iOS. In questo modo i clienti possono scegliere di includerla o escluderla in base alle loro esigenze di funzionalità. Includerla aumenta le dimensioni del pacchetto.

Le versioni vengono pubblicate tramite CocoaPods under the name`AmazonIVSBroadcast`. Aggiungere questa dipendenza al proprio Podfile:

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

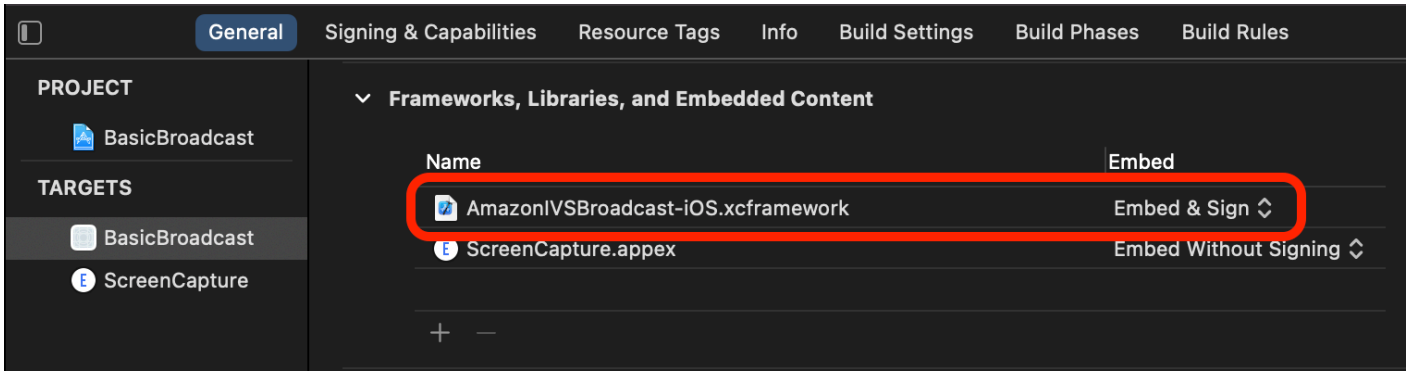
Eseguire `pod install` e l'SDK sarà disponibile nel `.xcworkspace`.

Importante: l'SDK di trasmissione per lo streaming in tempo reale IVS (ad esempio, con la sottospecifica della fase) include tutte le funzionalità dell'SDK di trasmissione per lo streaming a bassa latenza IVS. Non è possibile integrare entrambi gli SDK nello stesso progetto. Se aggiungi lo stage subspec via CocoaPods al tuo progetto, assicurati di rimuovere qualsiasi altra riga nel Podfile contenente. `AmazonIVSBroadcast` Ad esempio, non hai entrambe queste righe nel tuo Podfile:

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

Approccio alternativo: installare manualmente il framework

1. Scaricate l'ultima versione da https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip.
2. Estrarre i contenuti dell'archivio. `AmazonIVSBroadcast.xcframework` contiene l'SDK sia per il dispositivo sia per il simulatore.
3. Incorporare `AmazonIVSBroadcast.xcframework` trascinandolo nella sezione Framework, librerie e contenuto incorporato della scheda Generali per il target dell'applicazione.



Richiedere autorizzazioni

L'app deve richiedere l'autorizzazione per accedere alla fotocamera e al microfono dell'utente. (Questo non riguarda solo Amazon IVS, ma qualsiasi applicazione che abbia bisogno di accedere alle fotocamere e ai microfoni.)

Qui, controlliamo se l'utente ha già concesso le autorizzazioni e, in caso contrario, ne facciamo richiesta:

```
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.
}
```

È necessario eseguire questa operazione per entrambe le tipologie di contenuti multimediali `.video` e `.audio`, se si desidera accedere rispettivamente a fotocamere e microfoni.

Inoltre, è necessario aggiungere voci per `NSCameraUsageDescription` e `NSMicrophoneUsageDescription` a `Info.plist`. In caso contrario, quando si tenta di richiedere le autorizzazioni l'app potrebbe subire un arresto anomalo.

Disabilitare il timer di inattività dell'applicazione

Questo passaggio è facoltativo, ma è consigliato. Impedisce al dispositivo di andare in sospensione durante l'utilizzo dell'SDK di trasmissione, che causerebbe l'interruzione della trasmissione.

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

Pubblicazione e sottoscrizione

Concetti

La funzionalità in tempo reale si basa su tre concetti fondamentali: fase, strategia e renderer. L'obiettivo di progettazione è ridurre al minimo la quantità di logica lato client necessaria per creare un prodotto funzionante.

Stage

La classe `IVSStage` è il principale punto di interazione tra l'applicazione host e l'SDK. La classe rappresenta lo stage stesso e serve per entrare e uscire dallo stage. La creazione o la partecipazione

a uno stage richiedono una stringa di token valida e non scaduta dal piano di controllo (control-plane) (rappresentata come `token`). Entrare e uscire da uno stage è semplice.

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

try stage.join()

stage.leave()
```

La classe `IVSStage` è anche il luogo in cui possono essere collegati `IVSStageRenderer` e `IVSErrorDelegate`:

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

Strategia

Il protocollo `IVSStageStrategy` consente all'applicazione host di comunicare lo stato desiderato dello stage all'SDK. È necessario implementare tre funzioni: `shouldSubscribeToParticipant`, `shouldPublishParticipant` e `streamsToPublishForParticipant`. Sono tutte analizzate di seguito.

Sottoscrizione ai partecipanti

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

Quando un partecipante remoto partecipa a uno stage, l'SDK interroga l'applicazione host sullo stato della sottoscrizione desiderato per quel partecipante. Le opzioni sono `.none`, `.audioOnly` e `.audioVideo`. Quando si restituisce un valore per questa funzione, l'applicazione host non deve preoccuparsi dello stato di pubblicazione, dello stato della sottoscrizione corrente o dello stato della connessione allo stage. Se viene restituito `.audioVideo`, l'SDK attende che il partecipante remoto effettui la pubblicazione prima della sottoscrizione e aggiorna l'applicazione host tramite il renderer durante tutto il processo.

Di seguito è riportata un'implementazione di esempio:

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

```
    return .audioVideo
}
```

Questa è l'implementazione completa di questa funzione per un'applicazione host che vuole sempre che tutti i partecipanti si vedano tra loro, ad esempio un'applicazione di chat video.

Sono possibili anche implementazioni più avanzate. Utilizza la proprietà `attributes` su `IVSParticipantInfo` per iscriverti selettivamente ai partecipanti in base agli attributi forniti dal server:

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

Questa può essere usata per creare uno stage in cui i moderatori possano monitorare tutti gli ospiti senza essere visti o ascoltati. L'applicazione host potrebbe utilizzare una logica aziendale aggiuntiva per consentire ai moderatori di vedersi ma rimanendo invisibili agli ospiti.

Pubblicazione

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

Una volta connesso allo stage, l'SDK interroga l'applicazione host per vedere se un dato partecipante deve eseguire una pubblicazione. Viene richiamata solo per i partecipanti locali che hanno il permesso di pubblicare in base al token fornito.

Di seguito è riportata un'implementazione di esempio:

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

Si tratta di un'applicazione di chat video standard in cui gli utenti vogliono sempre pubblicare. Possono disattivare e riattivare l'audio e il video per essere nascosti o visti/ascoltati immediatamente.

Possono anche usare il comando di pubblicazione/annullamento della pubblicazione, ma è molto più lento. È preferibile disattivare/riattivare l'audio nei casi d'uso in cui è consigliabile modificare spesso la visibilità.

Scelta dei flussi da pubblicare

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

Durante la pubblicazione, serve a determinare quali flussi audio e video devono essere pubblicati. Questo argomento verrà trattato dettagliatamente più avanti in Pubblicazione di un flusso multimediale.

Aggiornamento della strategia

La strategia è pensata per essere dinamica: è possibile modificare i valori restituiti da una qualsiasi delle funzioni precedenti in qualsiasi momento. Ad esempio, se l'applicazione host non desidera pubblicare finché l'utente finale non tocca un pulsante, è possibile restituire una variabile da `shouldPublishParticipant` (del tipo `hasUserTappedPublishButton`). Quando quella variabile cambia in base a un'interazione da parte dell'utente finale, chiama `stage.refreshStrategy()` per segnalare all'SDK che dovrebbe eseguire una query sulla strategia per i valori più recenti, applicando solo quanto modificato. Se l'SDK rileva che il valore `shouldPublishParticipant` è cambiato, avvierà il processo di pubblicazione. Se le query dell'SDK e tutte le funzioni restituiscono lo stesso valore di prima, la chiamata `refreshStrategy` non apporterà alcuna modifica allo stage.

Se il valore restituito di `shouldSubscribeToParticipant` cambia da `.audioVideo` a `.audioOnly`, il flusso video verrà rimosso per tutti i partecipanti con valori restituiti modificati, se in precedenza esisteva un flusso video.

In genere, lo stage utilizza la strategia per applicare in modo più efficiente la differenza tra le strategie precedenti e quelle attuali, senza che l'applicazione host debba preoccuparsi di tutto lo stato necessario per gestirla correttamente. Per questo motivo, considera la chiamata a `stage.refreshStrategy()` come un'operazione a basso costo, perché non viene eseguita a meno che la strategia non cambi.

Renderer

Il protocollo `IVSStageRenderer` comunica lo stato desiderato dello stage all'applicazione host. Gli aggiornamenti all'interfaccia utente dell'applicazione host in genere possono essere alimentati interamente dagli eventi forniti dal renderer. Il renderer fornisce le funzioni seguenti:

```
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?)
```

Non è previsto che le informazioni fornite dal renderer influiscano sui valori restituiti della strategia. Ad esempio, non è previsto che il valore restituito di `shouldSubscribeToParticipant` cambi quando viene chiamato `participant:didChangePublishState`. Se l'applicazione host desidera effettuare la sottoscrizione a un particolare partecipante, deve restituire il tipo di abbonamento desiderato indipendentemente dallo stato di pubblicazione di quel partecipante. L'SDK è responsabile di garantire che lo stato desiderato della strategia venga applicato al momento giusto in base allo stato dello stage.

Ricorda che solo i partecipanti alla pubblicazione attivano `participantDidJoin` e ogni volta che un partecipante interrompe la pubblicazione o abbandona la sessione dello stage viene attivato `participantDidLeave`.

Pubblicazione di un flusso multimediale

I dispositivi locali, come microfoni e fotocamere integrati, vengono rilevati tramite `IVSDeviceDiscovery`. Ecco un esempio di selezione della fotocamera frontale e del microfono predefinito, che vengono poi restituiti come `IVSLocalStageStreams` per la pubblicazione dall'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]
}
```

Visualizzazione e rimozione dei partecipanti

Una volta completata la sottoscrizione, riceverai una serie di oggetti `IVSStageStream` tramite la funzione `didAddStreams` del renderer. Per visualizzare un'anteprima o ricevere le statistiche del livello audio su questo partecipante, puoi accedere all'oggetto `IVSDevice` sottostante dal flusso:

```
if let imageDevice = stream.device as? IVSImageDevice {
    let preview = imageDevice.previewView()
    /* attach this UIView subclass to your view */
} else if let audioDevice = stream.device as? IVSAudioDevice {
    audioDevice.setStatsCallback( { stats in
        /* process stats.peak and stats.rms */
    })
}
```

Quando un partecipante interrompe la pubblicazione o annulla l'iscrizione, la funzione `didRemoveStreams` viene chiamata con i flussi che sono stati rimossi. Le applicazioni host devono utilizzarlo come segnale per rimuovere il flusso video del partecipante dalla gerarchia delle visualizzazioni.

`didRemoveStreams` viene richiamato per tutti gli scenari in cui un flusso potrebbe essere rimosso, tra cui:

- Il partecipante remoto interrompe la pubblicazione.
- Un dispositivo locale annulla l'iscrizione o modifica l'abbonamento da `.audioVideo` a `.audioOnly`.
- Il partecipante remoto lascia lo stage.
- Il partecipante locale lascia lo stage.

Poiché `didRemoveStreams` viene richiamato per tutti gli scenari, non è richiesta alcuna logica aziendale personalizzata per la rimozione dei partecipanti dall'interfaccia utente durante le operazioni di abbandono remote o locali.

Disattivazione e riattivazione dell'audio dei flussi multimediali

Gli oggetti `IVSLocalStageStream` hanno una funzione `setMuted` che controlla se l'audio del flusso è disattivato. Questa funzione può essere richiamata sul flusso prima o dopo la restituzione dalla funzione della strategia `streamsToPublishForParticipant`.

Importante: se una nuova istanza di oggetto `IVSLocalStageStream` viene restituita da `streamsToPublishForParticipant` dopo una chiamata a `refreshStrategy`, lo stato di silenziamento del nuovo oggetto di flusso viene applicato allo stage. Fai attenzione quando crei nuove istanze `IVSLocalStageStream` per assicurarti che lo stato di silenziamento previsto venga mantenuto.

Monitoraggio dello stato di silenziamento dei contenuti multimediali dei partecipanti remoti

Quando un partecipante modifica lo stato di silenziamento del proprio flusso video o audio, la funzione `didChangeMutedStreams` del renderer viene richiamata con una serie di flussi che sono stati modificati. Usa la proprietà `isMuted` su `IVSStageStream` per aggiornare l'interfaccia utente di conseguenza:

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

Creazione di una configurazione dello stage

Per personalizzare i valori della configurazione video di uno stage, usa `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
```

Ottenimento delle statistiche WebRTC

Per ottenere le statistiche WebRTC più recenti per un flusso di pubblicazione o un flusso di iscrizione, usa `requestRTCStats` su `IVSStageStream`. Quando una raccolta è completata, riceverai statistiche tramite il `IVSStageStreamDelegate` che può essere impostato su `IVSStageStream`. Per raccogliere continuamente statistiche WebRTC, chiama questa funzione su 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)")
      }
   }
}
```

Ottieni gli attributi dei partecipanti

Se specifichi gli attributi nella richiesta dell'endpoint `CreateParticipantToken`, puoi visualizzarli nelle proprietà `IVSParticipantInfo`:

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

Continuazione della sessione in background

Quando l'app entra in background, puoi continuare a rimanere nello stage mentre ascolti l'audio remoto, anche se non puoi continuare a inviare la tua immagine e il tuo audio. Dovrai aggiornare la tua implementazione `IVSStrategy` per interrompere la pubblicazione e iscriverti a `.audioOnly` (o `.none`, se applicabile):

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

Quindi effettua una chiamata a `stage.refreshStrategy()`.

Abilitazione/disabilitazione della codifica a livelli con simulcast

Quando si pubblica un flusso multimediale, l'SDK trasmette flussi video di alta e bassa qualità, pertanto i partecipanti remoti possono sottoscrivere il flusso anche se hanno una larghezza di banda limitata per il downlink. La codifica a livelli con simulcast è attiva per impostazione predefinita. Puoi disabilitarla con `IVSSimulcastConfiguration`:

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

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

// Other Stage implementation code
```

Trasmissione della fase a un canale IVS

Per trasmettere uno stage, crea una `IVSBroadcastSession` separata e segui le normali istruzioni per la trasmissione con l'SDK descritte sopra. La proprietà `device` su `IVSStageStream` sarà un `IVSImageDevice` o `IVSAudioDevice` come mostrato nel frammento precedente; queste possono essere collegate al `IVSBroadcastSession.mixer` per trasmettere l'intero stage in un layout personalizzabile.

Facoltativamente, puoi comporre una fase e trasmetterla a un canale IVS a bassa latenza in modo da raggiungere un pubblico più vasto. Consulta Abilitazione di più host su un flusso Amazon IVS nella Guida per l'utente dello streaming a bassa latenza di IVS.

Come iOS sceglie la risoluzione della fotocamera e la frequenza dei fotogrammi

La telecamera gestita dall'SDK di trasmissione ottimizza la risoluzione e la frequenza dei fotogrammi (frames-per-secondo FPS) per ridurre al minimo la produzione di calore e il consumo di energia. Questa sezione spiega come vengono selezionati la risoluzione e la frequenza dei fotogrammi per ottimizzare le applicazioni host per i rispettivi casi d'uso.

Quando si crea un `IVSLocalStageStream` con una `IVSCamera`, la fotocamera è ottimizzata per una frequenza dei fotogrammi di `IVSLocalStageStreamVideoConfiguration.targetFramerate` e una risoluzione di `IVSLocalStageStreamVideoConfiguration.size`. La chiamata `IVSLocalStageStream.setConfiguration` aggiorna la fotocamera con i valori più recenti.

Anteprima della fotocamera

Se si crea un'anteprima di una `IVSCamera` senza collegarla a una `IVSBroadcastSession` o una `IVSStage`, per impostazione predefinita vengono impostate una risoluzione di 1080p e una frequenza dei fotogrammi di 60 FPS.

Trasmissione a una fase

Quando si utilizza una `IVSBroadcastSession` per trasmettere una `IVSStage`, l'SDK cerca di ottimizzare la telecamera con una risoluzione e una frequenza dei fotogrammi che soddisfino i criteri di entrambe le sessioni.

Ad esempio, se la configurazione di trasmissione è impostata per avere una frequenza dei fotogrammi di 15 FPS e una risoluzione di 1080p, mentre la fase ha una frequenza dei fotogrammi

di 30 FPS e una risoluzione di 720p, l'SDK selezionerà una configurazione della fotocamera con una frequenza dei fotogrammi di 30 FPS e una risoluzione di 1080p. La `IVSBroadcastSession` salterà un fotogramma ogni due proveniente dalla fotocamera e la `IVSStage` ridimensionerà l'immagine da 1080p a 720p.

Se un'applicazione host prevede di utilizzare insieme `IVSBroadcastSession` e `IVSStage` con una fotocamera, si consiglia di impostare sui medesimi valori le proprietà `targetFramerate` e `size` delle rispettive configurazioni. Una mancata corrispondenza potrebbe causare la riconfigurazione automatica della fotocamera durante l'acquisizione del video, causando un breve ritardo nella consegna degli elementi video.

Se per il caso d'uso dell'applicazione host non fosse possibile utilizzare valori identici, creare prima la videocamera di qualità superiore impedirà alla telecamera di riconfigurarsi quando viene aggiunta la sessione di qualità inferiore. Ad esempio, se si trasmette a 1080p e 30 FPS e poi si prende parte a una fase impostata su 720p e 30 FPS, la fotocamera non si riconfigurerà automaticamente e il video continuerà senza interruzioni. Questo perché 720p è inferiore o uguale a 1080p e 30 FPS è inferiore o uguale a 30 FPS.

Frequenza dei fotogrammi, risoluzioni e proporzioni arbitrari

La maggior parte dell'hardware della fotocamera può corrispondere esattamente ai formati più comuni, come 720p a 30 FPS o 1080p a 60 FPS. Tuttavia, non è possibile fornire una corrispondenza esatta con tutti i formati. L'SDK di trasmissione sceglie la configurazione della fotocamera in base alle seguenti regole (in ordine di priorità):

1. La larghezza e l'altezza della risoluzione sono maggiori o uguali alla risoluzione desiderata, ma larghezza e altezza sono le più piccole possibili nel rispetto di questo vincolo.
2. La frequenza dei fotogrammi è maggiore o uguale alla frequenza dei fotogrammi desiderata, ma la frequenza dei fotogrammi è la più bassa possibile nel rispetto di questo vincolo.
3. Le proporzioni corrispondono alle proporzioni desiderate.
4. Se esistono più formati corrispondenti, viene utilizzato il formato con il campo visivo più ampio.

Di seguito, sono riportati due esempi:

- L'applicazione host sta cercando di trasmettere in 4k a 120 FPS. La fotocamera selezionata supporta solo 4k a 60 FPS oppure 1080p a 120 FPS. Il formato selezionato sarà 4k a 60 FPS, poiché la regola di risoluzione ha una priorità maggiore rispetto alla regola della frequenza dei fotogrammi.

- È richiesta una risoluzione irregolare, 1910x1070. La fotocamera utilizzerà 1920x1080. Attenzione: scegliendo una risoluzione come 1921x1080, la fotocamera passerà alla successiva risoluzione disponibile (ad esempio 2592x1944), il che comporta una penalizzazione della CPU e della larghezza di banda della memoria.

E per quanto riguarda Android?

Android non regola la risoluzione o la frequenza dei fotogrammi all'istante come fa iOS, quindi ciò non influisce sull'SDK di trasmissione Android.

Problemi noti e soluzioni alternative

- La modifica del routing audio Bluetooth può essere imprevedibile. Se si connette un nuovo dispositivo a metà sessione, iOS potrebbe cambiare o meno il routing di input in modo automatico. Inoltre, non è possibile scegliere tra più auricolari Bluetooth collegati contemporaneamente. Ciò accade sia nelle normali sessioni di trasmissione che in quelle relative allo stage.

  Soluzione alternativa: se si prevede di utilizzare un auricolare Bluetooth, collegarlo prima di avviare la trasmissione o lo stage e lasciarlo connesso per tutta la durata della sessione.
- I partecipanti che utilizzano iPhone 14, iPhone 14 Plus, iPhone 14 Pro o iPhone 14 Pro Max potrebbero causare un problema di eco audio per gli altri partecipanti.

  Soluzione alternativa: i partecipanti che utilizzano i dispositivi interessati possono utilizzare le cuffie per evitare il problema dell'eco per gli altri partecipanti.
- Quando un partecipante accede con un token utilizzato da un altro partecipante, la prima connessione viene disconnessa senza un errore specifico.

  Soluzione alternativa: nessuna.
- Può verificarsi un problema raro per cui l'editore sta pubblicando, ma lo stato di pubblicazione che gli abbonati ricevono è `inactive`.

  Soluzione alternativa: prova a uscire e poi a partecipare alla sessione. Se il problema persiste, crea un nuovo token per l'editore.
- Quando un partecipante pubblica o si iscrive, è possibile ricevere un errore con il codice 1400 che indica la disconnessione dovuta a un problema di rete, anche quando la rete è stabile.

  Soluzione alternativa: prova a effettuare nuovamente la pubblicazione/sottoscrizione.

- Durante una sessione di stage può verificarsi un raro problema di distorsione audio a intermittenza, in genere durante le chiamate di lunga durata.

  Soluzione alternativa: il partecipante con audio distorto può uscire e accedere nuovamente alla sessione oppure annullare la pubblicazione e ripubblicare l'audio per risolvere il problema.

Gestione errori

Errori irreversibili e non irreversibili

L'oggetto errore ha un valore booleano "è irreversibile". Questa è una voce del dizionario sotto `IVSBroadcastErrorIsFatalKey` che contiene un valore booleano.

In generale, gli errori irreversibili sono legati alla connessione al server degli stage (ad es. non è possibile stabilire una connessione oppure la connessione viene interrotta e non può essere recuperata). L'applicazione dovrebbe ricreare lo stage e riaccedervi, possibilmente con un nuovo token o quando la connettività del dispositivo viene ripristinata.

Gli errori non irreversibili sono generalmente correlati allo stato di pubblicazione/sottoscrizione e vengono gestiti dall'SDK, che riprova l'operazione di pubblicazione/sottoscrizione.

Puoi controllare questa proprietà:

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

Errori di accesso

Token non conforme

Ciò accade quando il token dello stage non è conforme.

L'SDK genera un'eccezione Swift con codice di errore = 1000 e IVS = YES. BroadcastErrorIsFatalKey

Azione: crea un token valido e riprova a partecipare.

Token scaduto

Ciò accade quando il token dello stage è scaduto.

L'SDK genera un'eccezione Swift con codice di errore = 1001 e IVS = YES. BroadcastErrorIsFatalKey

Azione: crea un nuovo token e riprova a partecipare.

Token non valido o revocato

Ciò accade quando il token dello stage è conforme ma viene rifiutato dal server degli stage. Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK chiama `stage(didChange connectionState, withError error)` con codice di errore = 1026 e IVS = YES. BroadcastErrorIsFatalKey

Azione: crea un token valido e riprova a partecipare.

Errori di rete per l'accesso iniziale

Ciò accade quando l'SDK non riesce a contattare il server degli stage per stabilire una connessione. Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK chiama `stage(didChange connectionState, withError error)` con codice di errore = 1300 e IVS = YES. BroadcastErrorIsFatalKey

Azione: attendi il ripristino della connettività del dispositivo e riprova a connetterti.

Errori di rete quando è già stato effettuato l'accesso

Se la connessione di rete del dispositivo si interrompe, l'SDK potrebbe perdere la connessione ai server dello stage. Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK chiama `stage(didChange connectionState, withError error)` con codice di errore = 1300 e valore IVS = YES. BroadcastErrorIsFatalKey

Azione: attendi il ripristino della connettività del dispositivo e riprova a connetterti.

Errori di pubblicazione/sottoscrizione

Initial

Esistono diversi tipi di errori:

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

Questi vengono segnalati in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK riprova l'operazione per un numero limitato di volte. Durante i nuovi tentativi, lo stato di pubblicazione/sottoscrizione è `ATTEMPTING_PUBLISH`/`ATTEMPTING_SUBSCRIBE`. Se i nuovi tentativi hanno esito positivo, lo stato diventa `PUBLISHED`/`SUBSCRIBED`.

L'SDK chiama `IVSErrorSourceDelegate:didEmitError` con il codice di errore pertinente e IVS = NO. BroadcastErrorIsFatalKey

Azione: non è necessaria alcuna azione, poiché l'SDK riprova automaticamente. Facoltativamente, l'applicazione può aggiornare la strategia per imporre ulteriori tentativi.

Già stabilita, poi fallita

Una pubblicazione o una sottoscrizione possono fallire una volta stabilite, molto probabilmente a causa di un errore di rete. Il codice di errore per una "connessione peer interrotta a causa di un errore di rete" è 1400.

Ciò viene segnalato in modo asincrono tramite il renderer dello stage fornito dall'applicazione.

L'SDK riprova l'operazione di pubblicazione/sottoscrizione. Durante i nuovi tentativi, lo stato di pubblicazione/sottoscrizione è `ATTEMPTING_PUBLISH`/`ATTEMPTING_SUBSCRIBE`. Se i nuovi tentativi hanno esito positivo, lo stato diventa `PUBLISHED`/`SUBSCRIBED`.

L'SDK chiama `didEmitError` con codice di errore = 1400 e IVS = NO. BroadcastErrorIsFatalKey

Azione: non è necessaria alcuna azione, poiché l'SDK riprova automaticamente. Facoltativamente, l'applicazione può aggiornare la strategia per imporre ulteriori tentativi. In caso di perdita totale della connettività, è probabile che anche la connessione agli stage fallisca.

SDK di trasmissione IVS: origini di immagini personalizzate (streaming in tempo reale)

Le origini di input di immagini personalizzate consentono a un'applicazione di fornire il proprio input di immagini all'SDK di trasmissione anziché limitarsi alle fotocamere preimpostate. Una origine di immagine personalizzata può essere semplice come una filigrana semitrasparente o una scena statica "torno subito" oppure può consentire all'app di eseguire ulteriori elaborazioni personalizzate come l'aggiunta di filtri di bellezza alla fotocamera.

Quando si utilizza una sorgente di input di immagine personalizzata per il controllo personalizzato della fotocamera (ad esempio l'utilizzo di librerie di filtri estetici che richiedono l'accesso alla fotocamera), l'SDK di trasmissione non è più responsabile della gestione della fotocamera. Invece, l'applicazione è responsabile della corretta gestione del ciclo di vita della fotocamera. Consulta la documentazione ufficiale della piattaforma su come la tua applicazione dovrebbe gestire la fotocamera.

Android

Dopo aver creato una sessione `DeviceDiscovery`, crea un'origine di input di immagine:

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

Questo metodo restituisce un `CustomImageSource`, che è una sorgente immagine supportata da un Surface Android standard. La classe secondaria `SurfaceSource` può essere ridimensionata e ruotata. Puoi inoltre creare un `ImagePreviewView` per visualizzare un'anteprima del contenuto.

Per recuperare il `Surface` sottostante:

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

Questo `Surface` può essere utilizzato come buffer di output per producer di immagini come Camera2, OpenGL ES e altre librerie. Il caso d'uso più semplice è disegnare direttamente una bitmap statica o un colore sulla tela di Surface. Tuttavia, molte librerie (come le librerie di filtri estetici) forniscono un metodo che consente a un'applicazione di specificare un `Surface` esterno per il rendering. È possibile utilizzare un metodo del genere per passare questo `Surface` alla libreria di filtri, il che consente alla libreria di emettere frame elaborati per lo streaming della sessione di trasmissione.

Questa `CustomImageSource` può essere avvolta in un `LocalStageStream` e restituito dal `StageStrategy` per la pubblicazione su un `Stage`.

iOS

Dopo aver creato una sessione `DeviceDiscovery`, crea un'origine di input di immagine:

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

Questo metodo restituisce un `IVSCustomImageSource`, che è una fonte di immagini che consente alla domanda di inviare `CMSampleBuffers` manualmente. Per i formati pixel supportati, consulta Riferimento all'SDK di trasmissione iOS; un collegamento alla versione più recente è presente nel manuale Note di rilascio di Amazon IVS per l'ultima versione dell'SDK di trasmissione.

I campioni inviati all'origine personalizzata verranno trasmessi alla fase:

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

Per lo streaming di video, utilizzare questo metodo in una richiamata. Ad esempio, se si utilizza la fotocamera, ogni volta che viene ricevuto un nuovo buffer campione da un `AVCaptureSession`, l'applicazione può inoltrare il buffer campione alla sorgente di immagine personalizzata. Se lo desideri, l'applicazione può applicare ulteriori elaborazioni (come un filtro di bellezza) prima di inviare il campione alla sorgente di immagine personalizzata.

La `IVSCustomImageSource` può essere avvolta in un `IVSLocalStageStream` e restituito dal `IVSStageStrategy` per la pubblicazione su un `Stage`.

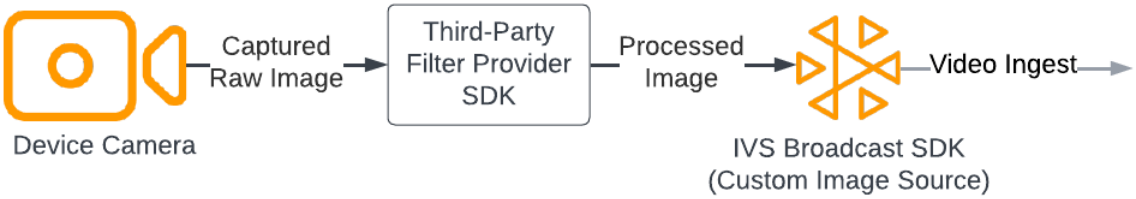
SDK di trasmissione IVS: filtri di fotocamere di terze parti (streaming in tempo reale)

Questa guida presuppone che tu abbia già familiarità con le origini di immagini personalizzate e con l'integrazione dell'SDK di trasmissione in streaming in tempo reale IVS nella tua applicazione.

I filtri della fotocamera consentono ai creatori di streaming live di aumentare o modificare l'aspetto del viso o dello sfondo. Ciò può potenzialmente aumentare il coinvolgimento degli spettatori, attirare gli spettatori e migliorare l'esperienza di streaming live.

Integrazione di filtri di fotocamere di terze parti

Puoi integrare gli SDK dei filtri di fotocamere di terze parti con l'SDK di trasmissione IVS inviando l'output dell'SDK del filtro a una sorgente di input di immagini personalizzata. Le origini di input di immagini personalizzate consentono a un'applicazione di fornire il proprio input di immagini all'SDK di trasmissione anziché limitarsi alle fotocamere preimpostate. L'SDK di un fornitore di filtri di terze parti può gestire il ciclo di vita della fotocamera per elaborare le immagini dalla fotocamera, applicare un effetto di filtro e inviarle in un formato che può essere passato a una sorgente di immagini personalizzata.



Consulta la documentazione del tuo fornitore di filtri di terze parti per conoscere i metodi integrati per convertire un frame di una fotocamera, con l'effetto filtro, applicato a un formato che può essere passato a una sorgente di input di immagini personalizzata. Il processo varia a seconda della versione dell'SDK di trasmissione IVS utilizzata:

- Web: il fornitore del filtro deve essere in grado di eseguire il rendering del proprio output su un elemento dell'area di lavoro. Il metodo captureStream può quindi essere utilizzato per restituire un MediaStream dei contenuti dell'area di lavoro. MediaStream può quindi essere convertito in un'istanza di LocalStageStream e pubblicato su una fase.
- Android: l'SDK del fornitore del filtro può eseguire il rendering di un frame su un dispositivo Android `Surface` fornito dall'SDK di trasmissione IVS o convertire il frame in una bitmap. Se si utilizza una bitmap, è possibile renderizzarla sul `Surface` sottostante fornito dalla sorgente dell'immagine personalizzata, sbloccandola e scrivendola nell'area di lavoro.
- iOS: l'SDK di un fornitore di filtri di terze parti deve fornire un frame della fotocamera con un effetto filtro applicato come `CMSampleBuffer`. Per informazioni su come ottenere un `CMSampleBuffer` come risultato finale dopo l'elaborazione dell'immagine di una fotocamera, consulta la documentazione dell'SDK del fornitore di filtri di terze parti.

BytePlus

Android

Installazione e configurazione dell'SDK BytePlus Effects

Consulta la Guida per l'accesso di Android di BytePlus per i dettagli su come installare, inizializzare e configurare l'SDK BytePlus Effects.

Configurazione della sorgente di immagini personalizzata

Dopo aver inizializzato l'SDK, alimenta i fotogrammi della fotocamera elaborati con un effetto filtro applicato a una sorgente di input di immagini personalizzata. A tale scopo, crea un'istanza di un oggetto `DeviceDiscovery` e crea una sorgente di immagini personalizzata. Quando si utilizza una sorgente di input di immagini personalizzate per il controllo personalizzato della fotocamera, l'SDK di trasmissione non è più responsabile della gestione della fotocamera. Invece, l'applicazione è responsabile della corretta gestione del ciclo di vita della fotocamera.

Java

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

Converti l'output in una bitmap e lo invia a una sorgente di input di immagini personalizzata

Per consentire ai fotogrammi della fotocamera con un effetto filtro applicato dall'SDK BytePlus Effects di essere inoltrati direttamente all'SDK di trasmissione IVS, converti l'output di una texture dell'SDK di BytePlus Effects in una bitmap. Quando un'immagine viene elaborata, il metodo `onDrawFrame()` viene richiamato dall'SDK. Il metodo `onDrawFrame()` è un metodo pubblico dell'interfaccia GLSurfaceView.Renderer di Android. Nell'app di esempio per Android fornita da BytePlus, questo metodo viene richiamato su ogni fotogramma della fotocamera e genera una texture. Allo stesso tempo, puoi integrare il metodo `onDrawFrame()` con la logica per convertire questa texture in una bitmap e inviarla a una sorgente di input di immagini personalizzata. Come illustrato nel seguente esempio di codice, utilizza il metodo `transferTextureToBitmap` fornito dall'SDK BytePlus per eseguire questa conversione. Questo metodo è fornito dalla

libreria com.bytedance.labcv.core.util.ImageUtil dell'SDK BytePlus Effects, come illustrato nel seguente esempio di codice. È quindi possibile eseguire il rendering sull'Android `Surface` di un `CustomImageSource` sottostante scrivendo la bitmap risultante nell'area di lavoro di Surface. Numerose invocazioni successive dei risultati `onDrawFrame()` in una sequenza di bitmap e, se combinate, creano un flusso video.

Java

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


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

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

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

DeepAR

Android

Consulta la Guida all'integrazione Android di DeepAR per i dettagli su come integrare l'SDK DeepAR con l'SDK di trasmissione IVS Android.

iOS

Consulta la Guida all'integrazione iOS di DeepAR per i dettagli su come integrare l'SDK DeepAR con l'SDK di trasmissione IVS iOS.

Aggancia

App

Questa sezione presuppone che tu abbia già dimestichezza con la pubblicazione e la sottoscrizione di video utilizzando l'SDK di trasmissione Web.

Per integrare l'SDK Camera Kit di Snap con l'SDK di trasmissione Web per lo streaming in tempo reale IVS, è necessario:

1. Installazione dell'SDK Camera Kit e Webpack. (Il nostro esempio utilizza Webpack come bundler, ma puoi utilizzare qualsiasi bundler di tua scelta.)
2. Creare il `index.html`.
3. Aggiungi gli elementi di configurazione.
4. Visualizza e configura i partecipanti.
5. Visualizza le fotocamere e i microfoni collegati.
6. Crea una sessione Camera Kit.
7. Recupera e applica un obiettivo.
8. Trasforma l'output da una sessione di Camera Kit in un'area di lavoro.
9. Fornisci a Camera Kit una origine multimediale per il rendering e la pubblicazione di un `LocalStageStream`.
10. Crea un file di configurazione di Webpack.

Di seguito è riportata una descrizione di ciascuna di queste fasi.

Installazione dell'SDK Camera Kit e Webpack

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

Creazione di index.html

Quindi, create il boilerplate HTML e importa l'SDK di trasmissione Web come tag di script. Nel codice seguente, assicurati di sostituire `<SDK version>` con la versione dell'SDK di trasmissione che stai utilizzando.

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>
```

Aggiunta degli elementi di configurazione

Crea il codice HTML per selezionare una fotocamera e un microfono e specifica un token di partecipazione:

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>
```

Aggiungi codice HTML aggiuntivo al di sotto per visualizzare i feed delle fotocamere dei partecipanti locali e remoti:

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>
```

Carica la logica aggiuntiva, inclusi i metodi helper per configurare la fotocamera e il file JavaScript in bundle. (Più avanti in questa sezione, creerai questi file JavaScript e li raggrupperai in un unico file, in modo da poter importare Camera Kit come modulo. Il file JavaScript fornito in bundle conterrà la logica per configurare Camera Kit, applicare un obiettivo e pubblicare il feed della fotocamera con un obiettivo applicato a una fase.)

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>
```

Visualizzazione e configurazione dei partecipanti

Successivamente, crea `helpers.js`, che contiene i metodi helper che utilizzerai per visualizzare e configurare i partecipanti:

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;
}
```

Visualizzazione delle fotocamere e i microfoni collegati

Successivamente, crea `media-devices.js`, che contiene metodi helper per la visualizzazione di fotocamere e microfoni collegati al dispositivo:

JavaScript

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

/**
 * Returns an initial list of devices populated on the page selects
 */
async function initializeDeviceSelect() {
  const videoSelectEl = document.getElementById('video-devices');
  videoSelectEl.disabled = false;

  const { videoDevices, audioDevices } = await getDevices();
  videoDevices.forEach((device, index) => {
    videoSelectEl.options[index] = new Option(device.label, device.deviceId);
  });

  const audioSelectEl = document.getElementById('audio-devices');

  audioSelectEl.disabled = false;
  audioDevices.forEach((device, index) => {
    audioSelectEl.options[index] = new Option(device.label, device.deviceId);
  });
}

/**
 * Returns all devices available on the current device
```

```
 */
async function getDevices() {
  // Prevents issues on Safari/FF so devices are not blank
  await navigator.mediaDevices.getUserMedia({ video: true, audio: true });

  const devices = await navigator.mediaDevices.enumerateDevices();
  // Get all video devices
  const videoDevices = devices.filter((d) => d.kind === 'videoinput');
  if (!videoDevices.length) {
    console.error('No video devices found.');
  }

  // Get all audio devices
  const audioDevices = devices.filter((d) => d.kind === 'audioinput');
  if (!audioDevices.length) {
    console.error('No audio devices found.');
  }

  return { videoDevices, audioDevices };
}

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

async function getMic(deviceId) {
  return navigator.mediaDevices.getUserMedia({
    video: false,
    audio: {
      deviceId: deviceId ? { exact: deviceId } : null,
    },
  });
}
```

Creazione di una sessione Camera Kit

Crea `stages.js`, che contiene la logica per applicare un obiettivo al feed della fotocamera e pubblicare il feed in una fase. Nella prima parte di questo file, importiamo l'SDK di trasmissione e l'SDK Web di Camera Kit e inizializziamo le variabili che utilizzeremo con ciascun SDK. Creiamo una sessione Camera Kit chiamando `createSession` dopo aver avviato l'SDK Web di Camera Kit. Tieni presente che un oggetto elemento dell'area di lavoro viene passato a una sessione; ciò indica a Camera Kit di renderizzare in quell'area di lavoro.

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 });
```

Recupero e applicazione di un obiettivo

Per recuperare i tuoi obiettivi, inserisci il tuo ID gruppo obiettivi, che puoi trovare nel Portale per gli sviluppatori di Camera Kit. In questo esempio, manteniamo le cose semplici applicando il primo obiettivo nell'array Lens che viene restituito.

JavaScript

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

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

Trasforma l'output da una sessione di Camera Kit in un'area di lavoro

Utilizza il metodo captureStream per restituire un `MediaStream` dei contenuti dell'area di lavoro. L'area di lavoro conterrà un flusso video del feed della fotocamera con un obiettivo applicato. Inoltre, aggiungi ascoltatori di eventi per i pulsanti per disattivare l'audio della fotocamera e del microfono, nonché ascoltatori di eventi per entrare e uscire da una fase. Nell'ascoltatore di eventi per entrare a

far parte di una fase, passiamo a una sessione di Camera Kit e al `MediaStream` dall'area di lavoro in modo che possa essere pubblicata in una fase.

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();
  });
};
```

Fornire a Camera Kit un'origine multimediale per il rendering e un LocalStageStream

Per pubblicare un flusso video con un obiettivo applicato, crea una funzione chiamata `setCameraKitSource` da trasmettere nel `MediaStream` acquisito in precedenza dall'area di lavoro. Il `MediaStream` dall'area di lavoro non sta facendo nulla al momento perché non abbiamo ancora incorporato il feed della nostra fotcamera locale. Possiamo incorporare il nostro feed locale della fotocamera chiamando il metodo helper `getCamera` e assegnandolo a `localCamera`. Possiamo quindi passare il feed della fotocamera locale (via `localCamera`) e l'oggetto della sessione a `setCameraKitSource`. La funzione `setCameraKitSource` converte il feed locale della fotocamera in una sorgente di contenuti multimediali per CameraKit chiamando `createMediaStreamSource`. La sorgente multimediale di `CameraKit` viene quindi trasformata per rispecchiare la fotocamera frontale. L'effetto Obiettivo viene quindi applicato alla sorgente multimediale e renderizzato nell'area di lavoro output mediante una chiamata a `session.play()`.

Con l'obiettivo ora applicato all'immagine `MediaStream` catturata dall'area di lavoro, possiamo quindi procedere alla pubblicazione in una fase. Ciò è possibile creando un `LocalStageStream` con le tracce video tratte da `MediaStream`. Un'istanza di `LocalStageStream` può quindi essere passata a un `StageStrategy` per essere pubblicata.

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;
    },
  };
```

Il codice rimanente riportato di seguito serve per creare e gestire la nostra fase:

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();
```

Creazione di un file di configurazione di Webpack.

Crea `webpack.config.js` e aggiungi il seguente codice. Questo raggruppa la logica di cui sopra in modo da poter utilizzare l'istruzione import per utilizzare Camera Kit.

JavaScript

```
const path = require('path');
module.exports = {
```

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

Infine, esegui `npm run build` per raggruppare il tuo JavaScript come definito nel file di configurazione di Webpack. Potrai quindi utilizzare HTML e JavaScript da un server Web. Ad esempio, puoi usare il server HTTP di Python e aprire localhost:8000 per vedere il risultato:

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

Android

Per integrare l'SDK Camera Kit di Snap con l'SDK di trasmissione Android IVS, devi installare l'SDK Camera Kit, inizializzare una sessione Camera Kit, applicare un obiettivo e inviare l'output della sessione Camera Kit alla sorgente di input dell'immagine personalizzata.

Per installare l'SDK Camera Kit, aggiungi quanto segue al file `build.gradle` del modulo. Sostituisci `$cameraKitVersion` con l'ultima versione dell'SDK Camera Kit.

Java

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

Inizializza e ottieni un `cameraKitSession`. Camera Kit fornisce anche un comodo wrapper per le API CameraX di Android, quindi non è necessario scrivere una logica complicata per utilizzare CameraX con Camera Kit. È possibile utilizzare l'oggetto `CameraXImageProcessorSource` come sorgente per ImageProcessor, il che consente di avviare lo streaming di frame in anteprima dalla fotocamera.

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();
    }
```

Recupero e applicazione di un obiettivo

Puoi configurare gli obiettivi e il loro ordine nel carosello del Portale per sviluppatori di 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);
      }));
});
```

Per la trasmissione, invia i fotogrammi elaborati al `Surface` di base di una sorgente di immagini personalizzate. Usa un oggetto `DeviceDiscovery` e crea un oggetto `CustomImageSource` per restituire un `SurfaceSource`. È quindi possibile eseguire il rendering dell'output da una sessione `CameraKit` al `Surface` sottostante fornito da `SurfaceSource`.

Java

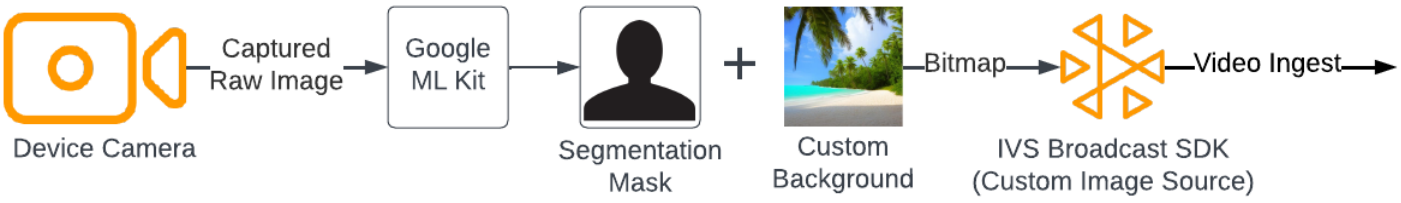
```
val publishStreams = ArrayList<LocalStageStream>()

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
```

Sostituzione dello sfondo

La sostituzione dello sfondo è un tipo di filtro della fotocamera che consente ai creatori di streaming live di modificare lo sfondo. Come illustrato nel seguente diagramma, la sostituzione dello sfondo comporta:

1. L'ottenimento di un'immagine della fotocamera dal feed live della fotocamera.
2. La segmentazione in componenti di primo piano e di secondo piano utilizzando Google ML Kit.
3. La combinazione della maschera di segmentazione risultante con un'immagine di sfondo personalizzata.
4. L'invio a una sorgente di immagini personalizzate per la trasmissione.



App

Questa sezione presuppone che tu abbia già dimestichezza con la pubblicazione e la sottoscrizione di video utilizzando l'SDK di trasmissione Web.

Per sostituire lo sfondo di uno streaming live con un'immagine personalizzata, utilizza il modello di segmentazione dei selfie con MediaPipe Image Segmenter. Si tratta di un modello di machine

learning che identifica quali pixel del fotogramma video sono in primo piano o sullo sfondo. Puoi quindi utilizzare i risultati del modello per sostituire lo sfondo di uno streaming live, copiando i pixel in primo piano dal feed video in un'immagine personalizzata che rappresenta il nuovo sfondo.

Per integrare la sostituzione dello sfondo con l'SDK di trasmissione Web per lo streaming in tempo reale IVS, è necessario:

1. Installare MediaPipe e Webpack. (Il nostro esempio utilizza Webpack come bundler, ma puoi utilizzare qualsiasi bundler di tua scelta.)
2. Creare il `index.html`.
3. Aggiungere elementi multimediali.
4. Aggiungere un tag di script.
5. Creare il `app.js`.
6. Caricare un'immagine di sfondo personalizzata.
7. Creare un'istanza di `ImageSegmenter`.
8. Eseguire il rendering del feed video su un'area di lavoro.
9. Creare la una logica di sostituzione dello sfondo.
10. Creare un file di configurazione di Webpack.
11. Raggruppare il file JavaScript.

Installazione di MediaPipe e Webpack

Per iniziare, installa i pacchetti npm `@mediapipe/tasks-vision` e `webpack`. L'esempio seguente utilizza Webpack come bundler JavaScript; se preferisci, puoi usare un bundler diverso.

JavaScript

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

Assicurati di aggiornare anche il tuo `package.json` per specificare `webpack` come script di compilazione:

JavaScript

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

```
  },
```

Creazione di index.html

Quindi, create il boilerplate HTML e importa l'SDK di trasmissione Web come tag di script. Nel codice seguente, assicurati di sostituire `<SDK version>` con la versione dell'SDK di trasmissione che stai utilizzando.

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>
```

Aggiunta di elementi multimediali

Quindi, aggiungi un elemento video e due elementi dell'area di lavoro all'interno del tag body. L'elemento video conterrà il feed live della fotocamera e sarà utilizzato come input per MediaPipe Image Segmenter. Il primo elemento dell'area di lavoro verrà utilizzato per renderizzare un'anteprima del feed che verrà trasmesso. Il secondo elemento dell'area di lavoro verrà utilizzato per renderizzare l'immagine personalizzata che verrà utilizzata come sfondo. Poiché la seconda area di lavoro con l'immagine personalizzata viene utilizzata solo come sorgente per copiare a livello di codice i pixel da essa all'area di lavoro finale, è nascosta alla vista.

JavaScript

```
<div class="row local-container">
```

```
      <video id="webcam" autoplay style="display: none"></video>
    </div>
    <div class="row local-container">
      <canvas id="canvas" width="640px" height="480px"></canvas>

      <div class="column" id="local-media"></div>
      <div class="static-controls hidden" id="local-controls">
        <button class="button" id="mic-control">Mute Mic</button>
        <button class="button" id="camera-control">Mute Camera</button>
      </div>
    </div>
    <div class="row local-container">
      <canvas id="background" width="640px" height="480px" style="display: none"></
canvas>
    </div>
```

Aggiunta di un tag di script

Aggiungi un tag di script per caricare un file JavaScript in bundle che conterrà il codice per eseguire la sostituzione dello sfondo e pubblicarlo su una fase:

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

Crea app.js

Quindi, crea un file JavaScript per ottenere gli oggetti elemento per gli elementi video e dell'area di lavoro creati nella pagina HTML. Importa i moduli `ImageSegmenter` e `FilesetResolver`. Il modulo `ImageSegmenter` verrà utilizzato per eseguire l'attività di segmentazione.

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";
```

Quindi, crea una funzione chiamata `init()` a recuperare il MediaStream dalla fotocamera dell'utente e richiama una funzione di callback ogni volta che un frame della fotocamera termina il caricamento. Aggiungi gli ascoltatori di eventi per i pulsanti per entrare e uscire da una fase.

Tieni presente che quando si entra in una fase, viene passata una variabile denominata `segmentationStream`. Si tratta di un flusso video catturato da un elemento dell'area di lavoro contenente un'immagine in primo piano sovrapposta all'immagine personalizzata che rappresenta lo sfondo. Successivamente, questo flusso personalizzato verrà utilizzato per creare un'istanza di un `LocalStageStream`, che può essere pubblicata in una fase.

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();
  });
};
```

Caricamento di un'immagine di sfondo personalizzata

Nella parte inferiore della funzione `init`, aggiungi il codice per chiamare una funzione denominata `initBackgroundCanvas`, che carica un'immagine personalizzata da un file locale e la rende su un'area di lavoro. Definiremo questa funzione nel passaggio successivo. Assegna il file `MediaStream` recuperato dalla fotocamera dell'utente all'oggetto video. Successivamente, questo oggetto video verrà passato a Image Segmenter. Inoltre, imposta una funzione denominata

`renderVideoToCanvas` come funzione di callback da richiamare ogni volta che un fotogramma video ha terminato il caricamento. Definiremo questa funzione in un passaggio successivo.

JavaScript

```
initBackgroundCanvas();

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

Implementiamo la funzione `initBackgroundCanvas`, che carica un'immagine da un file locale. In questo esempio, viene utilizzata l'immagine di una spiaggia come sfondo personalizzato. L'area di lavoro contenente l'immagine personalizzata verrà nascosta alla visualizzazione, poiché la unirai ai pixel di primo piano dell'elemento dell'area di lavoro contenente il feed della fotocamera.

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);
  };
};
```

Creazione di un'istanza di ImageSegmenter

Quindi, crea un'istanza di `ImageSegmenter`, che segmenterà l'immagine e restituirà il risultato come maschera. Quando crei un'istanza di un `ImageSegmenter`, utilizzerai il modello di segmentazione dei 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,
  });
};
```

Rendering del feed video su un'area di lavoro

Quindi, crea la funzione che esegue il rendering del feed video sull'altro elemento dell'area di lavoro. È necessario renderizzare il feed video su un'area di lavoro in modo da poter estrarre i pixel di primo piano da esso utilizzando l'API Canvas 2D. Durante questa operazione, passeremo anche un fotogramma video alla nostra istanza di `ImageSegmenter`, utilizzando il metodo segmentforVideo per segmentare il primo piano dallo sfondo nel fotogramma video. Quando il metodo segmentforVideo viene completato, richiama la nostra funzione di callback personalizzata, `replaceBackground`, per eseguire la sostituzione dello sfondo.

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);
};
```

Creazione di una logica di sostituzione dello sfondo

Crea la funzione `replaceBackground`, che unisce l'immagine di sfondo personalizzata con il primo piano dal feed della fotocamera per sostituire lo sfondo. La funzione recupera innanzitutto i dati dei

pixel sottostanti dell'immagine di sfondo personalizzata e del feed video dai due elementi dell'area di lavoro creati in precedenza. Quindi scorre attraverso la maschera fornita da `ImageSegmenter`, che indica quali pixel sono in primo piano. Mentre itera attraverso la maschera, copia selettivamente i pixel che contengono il feed della fotocamera dell'utente nei dati dei pixel di sfondo corrispondenti. Fatto ciò, converte i dati finali dei pixel con il primo piano copiato sullo sfondo e li disegna su un'area di lavoro.

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);
}
```

Per riferimento, ecco il file `app.js` completo contenente tutta la logica di cui sopra:

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();
```

Creazione di un file di configurazione di Webpack.

Aggiungi questa configurazione al file di configurazione di Webpack per raggruppare `app.js`, in modo che le chiamate di importazione funzionino:

JavaScript

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

Raggruppamento dei tuoi file JavaScript

```
npm run build
```

Avvia un semplice server HTTP dalla directory contenente `index.html` e apri `localhost:8000` per vedere il risultato:

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

Android

Per sostituire lo sfondo del tuo streaming live, puoi utilizzare l'API di segmentazione dei selfie di Google ML Kit. L'API di segmentazione dei selfie accetta l'immagine di una fotocamera come input

e restituisce una maschera che fornisce un punteggio di affidabilità per ogni pixel dell'immagine, indicando se era in primo piano o sullo sfondo. In base al punteggio di affidabilità, puoi quindi recuperare il colore dei pixel corrispondente dall'immagine di sfondo o dall'immagine in primo piano. Questo processo continua fino a quando non sono stati esaminati tutti i punteggi di affidabilità nella maschera. Il risultato è una nuova gamma di colori dei pixel contenenti pixel in primo piano combinati con i pixel dell'immagine di sfondo.

Per integrare la sostituzione dello sfondo con l'SDK di trasmissione per lo streaming in tempo reale IVS, è necessario:

1. Installare le librerie CameraX e Google ML Kit.
2. Inizializzare le variabili boilerplate.
3. Creare una sorgente di immagini personalizzate.
4. Gestire i frame della fotocamera.
5. Passre i frame della fotocamera a Google ML Kit.
6. Sovrapprre i frame della fotocamera in primo piano allo sfondo personalizzato.
7. Inserire la nuova immagine in una sorgente di immagini personalizzate.

Installazione delle librerie CameraX e di Google ML Kit

Per estrarre immagini dal feed live della fotocamera, utilizza la libreria CameraX di Android. Per installare l'SDK Camera Kit e Google ML Kit, aggiungi quanto segue al file `build.gradle` del modulo. Sostituisci `${camerax_version}` e `${google_ml_kit_version}` con la versione più recente delle librerie CameraX e Google ML Kit, rispettivamente.

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}"
```

Importa le seguenti librerie:

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
```

Inizializzazione delle variabili boilerplate.

Inizializza un'istanza di `ImageAnalysis` e un'istanza di un `ExecutorService`:

Java

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

Inizializza un'istanza Segmenter in STREAM_MODE:

Java

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

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

Creazione di una sorgente di immagini personalizzate

Nel metodo on`Create` della tua attività, crea un'istanza di un oggetto `DeviceDiscovery` e crea una sorgente di immagini personalizzate. Il `Surface` fornito dalla sorgente di immagini personalizzate riceverà l'immagine finale, con il primo piano sovrapposto a un'immagine di sfondo personalizzata. Creerai quindi un'istanza di un `ImageLocalStageStream` utilizzando la sorgente di immagini personalizzate. L'istanza di un `ImageLocalStageStream` (denominata `filterStream` in questo esempio) può quindi essere pubblicata in una fase. Consulta la Guida all'SDK di trasmissione Android IVS per le istruzioni di configurazione di una fase. Infine, crea anche un thread che verrà utilizzato per gestire la fotocamera.

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()
```

Gestione di frame della fotocamera

Quindi, crea una funzione per inizializzare la fotocamera. Questa funzione utilizza la libreria CameraX per estrarre immagini dal feed live della fotocamera. Innanzitutto, crea un'istanza di un `ProcessCameraProvider` chiamata `cameraProviderFuture`. Questo oggetto rappresenta un risultato futuro dell'ottenimento di un fornitore di fotocamere. Quindi carica un'immagine dal tuo progetto come bitmap. Questo esempio utilizza l'immagine di una spiaggia come sfondo, ma è possibile utilizzare qualsiasi immagine desiderata.

Quindi aggiungi un ascoltatore a `cameraProviderFuture`. Questo ascoltatore riceve una notifica quando la fotocamera diventa disponibile o se si verifica un errore durante il processo di ricerca di un fornitore di fotocamere.

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))
}
```

All'interno dell'ascoltatore, crea `ImageAnalysis.Builder` per accedere a ogni singolo fotogramma dal feed live della fotocamera. Imposta la strategia di contropressione su `STRATEGY_KEEP_ONLY_LATEST`. Ciò garantisce che per l'elaborazione venga fornito un solo fotogramma alla volta. Converte ogni singolo fotogramma della fotocamera in una bitmap in modo da poterne estrarre i pixel per combinarli successivamente con l'immagine di sfondo personalizzata.

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())
```

Passaggio dei frame della fotocamera a Google ML Kit

Quindi, crea un file `InputImage` e invialo all'istanza di Segmenter per l'elaborazione. Un `InputImage` può essere creato da un `ImageProxy` fornito dall'istanza di `ImageAnalysis`. Una volta fornito `InputImage` a Segmenter, viene restituita una maschera con punteggi di affidabilità che indicano la probabilità che un pixel sia in primo piano o sullo sfondo. Questa maschera fornisce anche proprietà di larghezza e altezza, che verranno utilizzate per creare un nuovo array contenente i pixel di sfondo dell'immagine di sfondo personalizzata caricata in precedenza.

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)
```

Sovrapposizione dei frame della fotocamera in primo piano allo sfondo personalizzato

Con la maschera contenente i punteggi di affidabilità, il frame della fotocamera come bitmap e i pixel a colori dell'immagine di sfondo personalizzata, hai tutto ciò di cui hai bisogno per sovrapporre il primo piano allo sfondo personalizzato. La funzione `overlayForeground` viene quindi chiamata con i parametri seguenti:

Java

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

Questa funzione scorre attraverso la maschera e verifica i valori di affidabilità per determinare se ottenere il colore dei pixel corrispondente dall'immagine di sfondo o dal frame della fotocamera. Se il valore di affidabilità indica che un pixel nella maschera è molto probabilmente sullo sfondo, otterrà il colore dei pixel corrispondente dall'immagine di sfondo; in caso contrario, otterrà il colore dei pixel corrispondente dal frame della fotocamera per costruire il primo piano. Una volta che la funzione termina l'iterazione nella maschera, viene creata e restituita una nuova bitmap utilizzando la nuova matrice di pixel colorati. Questa nuova bitmap contiene il primo piano sovrapposto allo sfondo personalizzato.

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
        )
```

```
    }
```

Inserimento della nuova immagine in una sorgente di immagini personalizzata

È quindi possibile scrivere la nuova bitmap nella `Surface` fornita dalla sorgente di immagini personalizzata. Questa operazione la trasmetterà alla fase.

Java

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

Ecco la funzione completa per ottenere i fotogrammi della fotocamera, passarli a Segmenter e sovrapporli allo sfondo:

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 di trasmissione IVS: modalità audio per dispositivi mobili (streaming in tempo reale)

La qualità audio è una parte importante di qualsiasi esperienza multimediale in team reali e non esiste una configurazione audio valida per tutti i casi d'uso. Per garantire ai tuoi utenti la migliore esperienza di ascolto di uno streaming IVS in tempo reale, i nostri SDK mobili offrono diverse configurazioni audio preimpostate oltre a personalizzazioni più avanzate, se necessarie.

Introduzione

Gli SDK di trasmissione mobile IVS forniscono una classe `StageAudioManager`. Questa classe è progettata per essere l'unico punto di contatto per il controllo delle modalità audio sottostanti su entrambe le piattaforme. Su Android, controlla AudioManager, che include la modalità audio, la sorgente audio, il tipo di contenuto, l'utilizzo e i dispositivi di comunicazione. Su iOS, controlla l'applicazione AVAudioSession e verifica che voiceProcessing sia abilitato.

Importante: non interagire con `AVAudioSession` o `AudioManager` direttamente mentre l'SDK di trasmissione in tempo reale IVS è attivo. Ciò potrebbe causare la perdita dell'audio o la registrazione o la riproduzione dell'audio sul dispositivo sbagliato.

Prima di creare il primo oggetto `DeviceDiscovery` o `Stage`, è necessario configurare 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
```

Se nulla è impostato su `StageAudioManager` prima dell'inizializzazione di un'istanza `DeviceDiscovery` o `Stage`, viene applicata automaticamente la preimpostazione `VideoChat`.

Preimpostazioni della modalità audio

L'SDK per la trasmissione in tempo reale offre tre preimpostazioni, ognuna personalizzata per i casi d'uso più comuni, come descritto di seguito. Per ogni preimpostazione, sono coperte cinque categorie principali che differenziano le preimpostazioni l'una dall'altra.

Videochat

Questa è la preimpostazione predefinita, progettata per quando il dispositivo locale deve avere una conversazione in tempo reale con altri partecipanti.

Categoria	Android	iOS
Cancellazione dell'eco	Abilitato	Abilitato
Controllo del volume	Volume delle chiamate	Volume delle chiamate
Selezione del microfono	Limitato in base al sistema operativo. I microfoni USB potrebbero non essere disponibili.	Limitato in base al sistema operativo. I microfoni USB potrebbero non essere disponibili. Gli auricolari Bluetooth che gestiscono contemporaneamente sia l'input che l'output (ad esempio gli AirPods) dovrebbero funzionare.
Uscita audio	Dovrebbe funzionare qualsiasi dispositivo di output.	Limitato in base al sistema operativo. Le cuffie con cavo potrebbero non essere disponibili.

Categoria	Android	iOS
Qualità audio	Medio/Basso. Suonerà come una telefonata, non come una riproduzi one multimediale.	Medio/Basso Suonerà come una telefonata, non come una riproduzi one multimediale.

Solo sottoscrizione

Questa preimpostazione è progettata per chi prevede di abbonarsi ad altri partecipanti alla pubblicazione ma non di pubblicare personalmente. Si concentra sulla qualità audio e supporta tutti i dispositivi di output disponibili.

Categoria	Android	iOS
Cancellazione dell'eco	Disabilitato	Disabilitato
Controllo del volume	Volume multimediale	Volume multimediale
Selezione del microfono	N/D, questa preimpostazione non è progettata per la pubblicazione.	N/D, questa preimpostazione non è progettata per la pubblicazione.
Uscita audio	Dovrebbe funzionare qualsiasi dispositivo di output.	Dovrebbe funzionare qualsiasi dispositivo di output.
Qualità audio	Elevato. Qualsiasi tipo di file multimediale dovrebbe essere chiaro, compresa la musica.	Elevato. Qualsiasi tipo di file multimediale dovrebbe essere chiaro, compresa la musica.

Studio

Questa preimpostazione è progettata per abbonamenti di alta qualità pur mantenendo la possibilità di pubblicazione. Richiede l'hardware di registrazione e riproduzione per consentire la cancellazione dell'eco. Un caso d'uso in questo caso potrebbe essere l'utilizzo di un microfono USB e un auricolare con cavo. L'SDK manterrà la massima qualità audio facendo affidamento sulla separazione fisica di tali dispositivi dalla causa dell'eco.

Categoria	Android	iOS
Cancellazione dell'eco	Disabilitato	Disabilitato
Controllo del volume	Volume multimediale, nella maggior parte dei casi. Volume di chiamata quando è collegato un microfono Bluetooth.	Volume multimediale
Selezione del microfono	Dovrebbe funzionare qualsiasi microfono.	Dovrebbe funzionare qualsiasi microfono.
Uscita audio	Dovrebbe funzionare qualsiasi dispositivo di output.	Dovrebbe funzionare qualsiasi dispositivo di output.
Qualità audio	Elevato. Entrambe le parti dovrebbero essere in grado di inviare musica e ascoltarla chiaramente dall'altra parte. Quando è collegato un auricolar e Bluetooth, la qualità audio diminuirà a causa dell'attivazione della modalità Bluetooth SCO.	Elevato. Entrambe le parti dovrebbero essere in grado di inviare musica e ascoltarla chiaramente dall'altra parte. Quando è collegato un auricolar e Bluetooth, la qualità audio potrebbe diminuire a causa dell'atti vazione della modalità Bluetooth SCO, a seconda dell'auricolare.

Casi d'uso avanzati

Oltre alle preimpostazioni, gli SDK di trasmissione in streaming in tempo reale iOS e Android consentono di configurare le modalità audio della piattaforma sottostante:

- Su Android, imposta AudioSource, Usage e ContentType.
- Su iOS, usa AVAudioSession.Category, AVAudioSession.CategoryOptions, AVAudioSession.Mode e la possibilità di attivare/disattivare se l'elaborazione vocale è abilitata o meno durante la pubblicazione.

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
```

Pubblicazione con Bluetooth su Android

L'SDK torna automaticamente alla preimpostazione `VIDEO_CHAT` su Android quando vengono soddisfatte le seguenti condizioni:

- La configurazione assegnata non utilizza il valore di utilizzo `VOICE_COMMUNICATION`.
- Un microfono Bluetooth è collegato al dispositivo.
- Il partecipante locale sta pubblicando in una fase.

Questa è una limitazione del sistema operativo Android per quanto riguarda l'utilizzo degli auricolari Bluetooth per la registrazione audio.

Integrazione con altri SDK

Poiché sia iOS che Android supportano solo una modalità audio attiva per applicazione, è normale che si verifichino conflitti se l'applicazione utilizza più SDK che richiedono il controllo della modalità audio. Quando si verificano questi conflitti, è possibile provare alcune strategie di risoluzione comuni, illustrate di seguito.

Corrisponde ai valori della modalità audio

Utilizzando le opzioni avanzate di configurazione audio dell'SDK IVS o le funzionalità degli altri SDK, fai in modo che i due SDK si allineino ai valori sottostanti.

Agora

iOS

Su iOS, dire all'SDK Agora di mantenere `AVAudioSession` attivo ne impedirà la disattivazione mentre l'SDK di trasmissione in streaming in tempo reale IVS lo utilizza.

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

Android

Evita di chiamare `setEnableSpeakerphone` su `RtcEngine` e chiama `enableLocalAudio(false)` durante la pubblicazione con l'SDK di trasmissione in streaming in tempo reale IVS. Potrai chiamare nuovamente `enableLocalAudio(true)` quando l'SDK IVS non è in fase di pubblicazione.

Utilizzo di Amazon EventBridge con lo streaming in tempo reale IVS

È possibile utilizzare Amazon EventBridge per monitorare i propri flussi Amazon Interactive Video Service (IVS).

Amazon IVS invia eventi di modifica relativi allo stato dei flussi ad Amazon EventBridge. Tutti gli eventi che vengono consegnati sono validi. Tuttavia, gli eventi vengono inviati sulla base del miglior tentativo il che significa che non vi è alcuna garanzia che:

- Gli eventi vengano consegnati: può verificarsi un determinato evento (ad esempio, un partecipante ha pubblicato qualcosa), ma è possibile che Amazon IVS non invii un evento di modifica corrispondente a EventBridge. Amazon IVS tenta di consegnare gli eventi per diverse ore prima di smettere.
- Gli eventi che vengono consegnati arriveranno in un periodo di tempo specificato: la ricezione di eventi è possibile fino a poche ore prima.
- Gli eventi vengono consegnati in ordine; gli eventi possono non essere ordinati, soprattutto se vengono inviati entro un breve periodo di tempo l'uno dall'altro. Ad esempio, potresti vedere il partecipante non pubblicato prima della pubblicazione effettiva del partecipante.

Sebbene sia raro che gli eventi siano mancanti, in ritardo o fuori sequenza, è consigliabile essere pronti a queste possibilità se si scrivono programmi business-critical che dipendono dall'ordine o dall'esistenza degli eventi di notifica.

È possibile creare regole EventBridge per qualsiasi evento tra quelli riportati di seguito.

Tipo di evento	Evento	Inviato quando…
Modifica dello stato di composizione dell'IVS	Errore destinazione	Un tentativo di invio a una destinazione non riuscito. Ad esempio, la trasmissione su un canale non è riuscita perché non c'era una chiave di streaming o era in corso un'altra trasmissione.

Tipo di evento	Evento	Inviato quando…
Modifica dello stato di composizione dell'IVS	Inizio destinazione	L'output verso una destinazione è stato avviato correttamente.
Modifica dello stato di composizione dell'IVS	Fine destinazione	L'output verso una destinazione è terminato.
Modifica dello stato di composizione dell'IVS	Riconnessione della destinazione	L'output verso una destinazione è stato interrotto ed è in corso un tentativo di riconnessione.
Modifica dello stato di composizione dell'IVS	Inizio sessione	È stata creata una sessione di composizione. Questo evento si attiva quando una pipeline del processo di composizione viene inizializzata correttamente. A questo punto, la pipeline di composizione è stata registrata correttamente per una fase, riceve contenuti multimediali ed è in grado di comporre video.
Modifica dello stato di composizione dell'IVS	Fine sessione	Una sessione di composizione è stata completata.
Modifica dello stato di composizione dell'IVS	Errore sessione	Impossibile inizializzare una pipeline di composizione a causa della mancata disponibilità delle risorse della fase o di qualsiasi altro errore interno.
Aggiornamento della fase IVS	Partecipante pubblicato	Un partecipante inizia a pubblicare in una fase.
Aggiornamento della fase IVS	Partecipante non pubblicato	Un partecipante ha interrotto la pubblicazione in una fase.

Creazione delle regole Amazon EventBridge per Amazon IVS

È possibile creare una regola che si attiva con un evento generato da Amazon IVS. Completa le operazioni riportate in Crea una regola in Amazon EventBridge nella Guida per l'utente di Amazon EventBridge. Quando si seleziona un servizio, seleziona Interactive Video Service (IVS).

Esempi: Modifica dello stato di composizione

Errore di destinazione: questo evento viene inviato quando un tentativo di output verso una destinazione non riesce. Ad esempio, la trasmissione su un canale non è riuscita perché non c'era una chiave di streaming o era in corso un'altra trasmissione.

```
{
   "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"
   }
}
```

Inizio destinazione: questo evento viene inviato quando l'output verso una destinazione è stato avviato correttamente.

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

Fine destinazione: questo evento viene inviato quando l'output verso una destinazione è terminato.

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

Riconnessione della destinazione: questo evento viene inviato quando l'output verso una destinazione è stato interrotto e viene tentata una riconnessione.

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

Inizio sessione: questo evento viene inviato quando è stata creata una sessione di composizione. Questo evento si attiva quando una pipeline del processo di composizione viene inizializzata correttamente. A questo punto, la pipeline di composizione è stata registrata correttamente per una fase, riceve contenuti multimediali ed è in grado di comporre video.

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

Fine sessione: questo evento viene inviato quando viene completata una sessione di composizione e tutte le risorse sono state eliminate.

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

Errore della sessione: questo evento viene inviato quando non è possibile inizializzare una pipeline di composizione a causa della mancata disponibilità delle risorse della fase, dell'assenza di partecipanti alla fase o di qualsiasi altro errore interno.

```
{
   "version": "0",
   "id": "01234567-0123-0123-0123-012345678901",
   "detail-type": "IVS Composition State Change",
   "source": "aws.ivs",
   "account": "aws_account_id",
   "time": "2017-06-12T10:23:43Z",
   "region": "us-east-1",
   "resources": [
     "arn:aws:ivs:us-east-1:aws_account_id:composition/123456789012"
   ],
   "detail": {
     "event_name": "Session Failure",
     "stage_arn": "<stage-arn>",
     "reason": "eg. no participants in the stage"
   }
}
```

Esempi: aggiornamenti della fase

Gli eventi di aggiornamento della fase includono un nome dell'evento (che classifica l'evento) e i metadati relativi all'evento. I metadati includono l'ID del partecipante che ha attivato l'evento, gli ID di fase e sessione associati e l'ID utente.

Partecipante pubblicato: questo evento viene inviato quando un partecipante inizia la pubblicazione in una fase.

```
{
```

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

Partecipante non pubblicato: questo evento viene inviato quando un partecipante ha interrotto la pubblicazione in una fase.

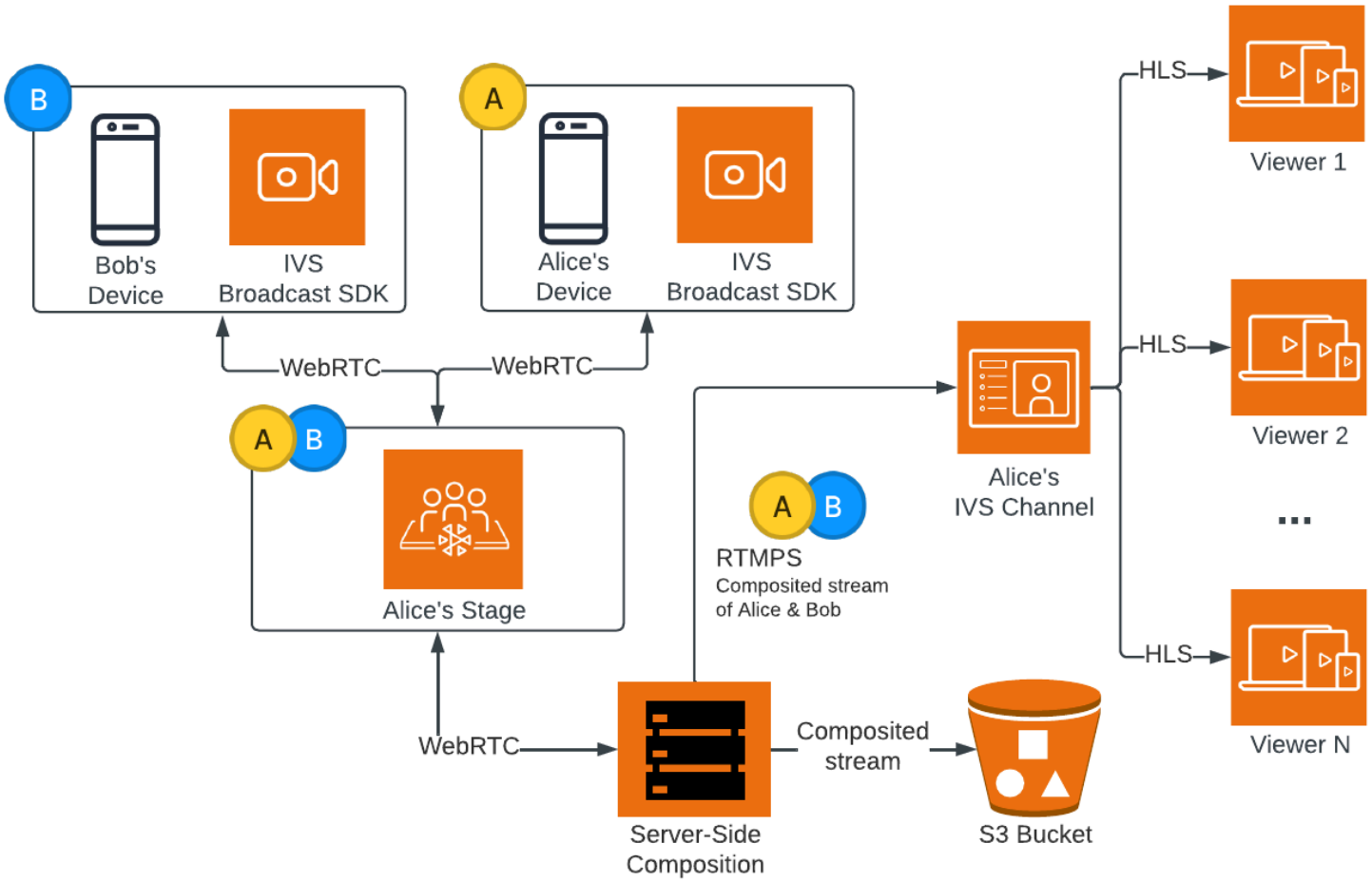
```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"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

Composizione lato server (streaming in tempo reale)

La composizione lato server utilizza un server IVS per mixare audio e video di tutti i partecipanti alla fase e quindi invia questo video misto a un canale IVS (ad esempio, per raggiungere un pubblico più ampio) o a un bucket S3. La composizione lato server viene richiamata tramite gli endpoint del piano di controllo (control-plane) IVS nella regione di origine della fase.

La trasmissione o la registrazione di una fase tramite la composizione lato server offrono numerosi vantaggi, rendendole scelte interessanti per gli utenti che cercano uno streaming live efficiente e affidabile.

Questo diagramma illustra come funziona la composizione lato server:



Vantaggi

Rispetto alla composizione lato client, la composizione lato server presenta i seguenti vantaggi:

- Carico client ridotto: con la composizione lato server, l'onere dell'elaborazione e della combinazione di sorgenti audio e video viene spostato dai singoli dispositivi client al server stesso. La composizione lato server elimina la necessità per i dispositivi client di utilizzare la CPU e le risorse di rete per comporre la vista e trasmetterla a IVS. Ciò significa che gli spettatori possono guardare la trasmissione senza che i loro dispositivi debbano gestire attività che richiedono molte risorse, il che può portare a una maggiore durata della batteria e a esperienze di visualizzazione più fluide.
- Qualità costante: la composizione lato server consente un controllo preciso sulla qualità, la risoluzione e il bitrate dello streaming finale. Ciò garantisce un'esperienza di visualizzazione coerente per tutti gli spettatori, indipendentemente dalle funzionalità dei singoli dispositivi.
- Resilienza: centralizzando il processo di composizione sul server, la trasmissione diventa più solida. Anche se per un editore si presentano limitazioni o fluttuazioni tecniche, il server può adattarsi e fornire uno streaming più fluido a tutto il pubblico.
- Efficienza della larghezza di banda: poiché il server gestisce la composizione, gli editori della fase non devono spendere una larghezza di banda aggiuntiva per trasmettere il video a IVS.

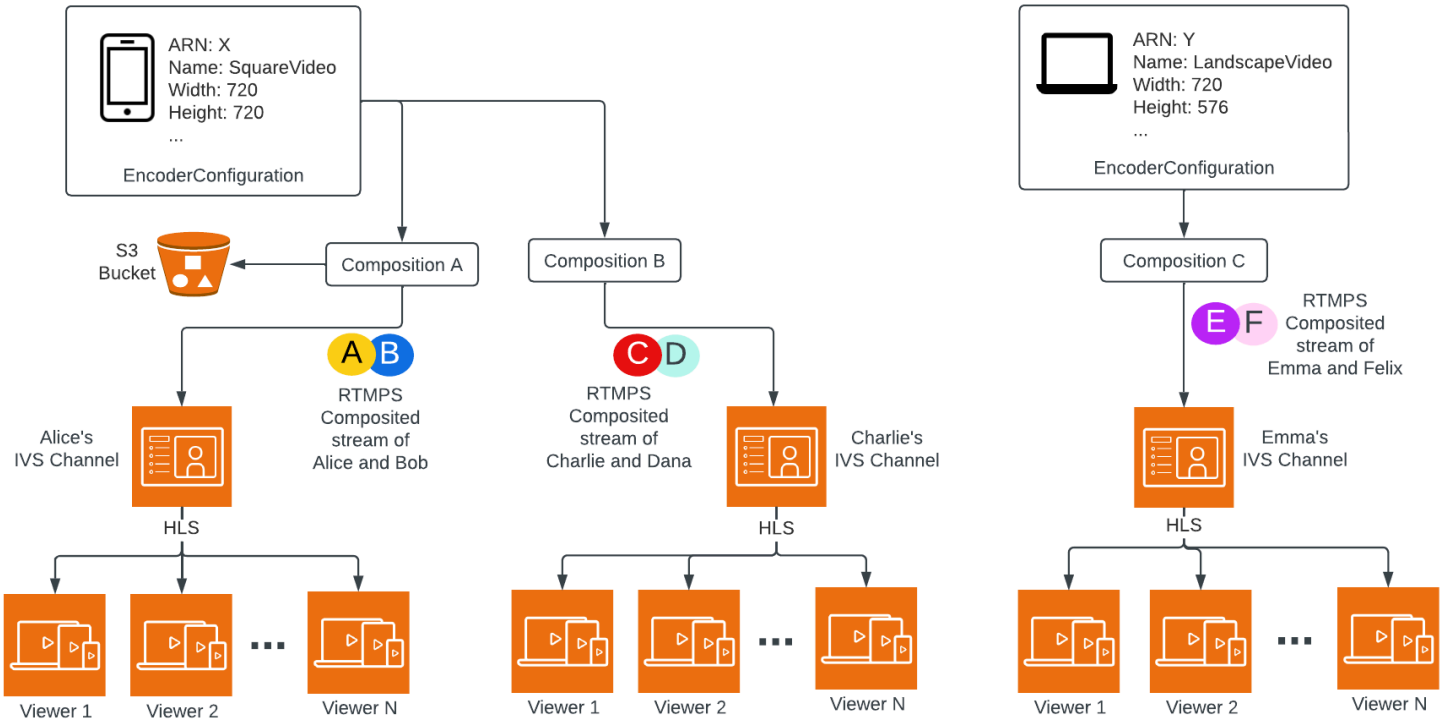
In alternativa, per trasmettere una fase su un canale IVS, puoi eseguire la composizione lato client; consulta Abilitazione di più host su un flusso IVS nella Guida per l'utente dello streaming a bassa latenza di IVS.

API IVS

La composizione lato server utilizza questi elementi chiave dell'API:

- Un oggetto EncoderConfiguration consente di personalizzare il formato del video da generare (altezza, larghezza, bitrate e altri parametri di streaming). È possibile riutilizzare un EncoderConfiguration ogni volta che si chiama l'endpoint StartComposition.
- Gli endpoint di composizione tengono traccia della composizione video e la trasmettono su un canale IVS.
- StorageConfiguration tiene traccia del bucket S3 in cui vengono registrate le composizioni.

Per utilizzare la composizione lato server, è necessario creare un EncoderConfiguration e collegarlo quando si chiama l'endpoint StartComposition. In questo esempio, SquareVideo EncoderConfiguration viene utilizzato in due composizioni:

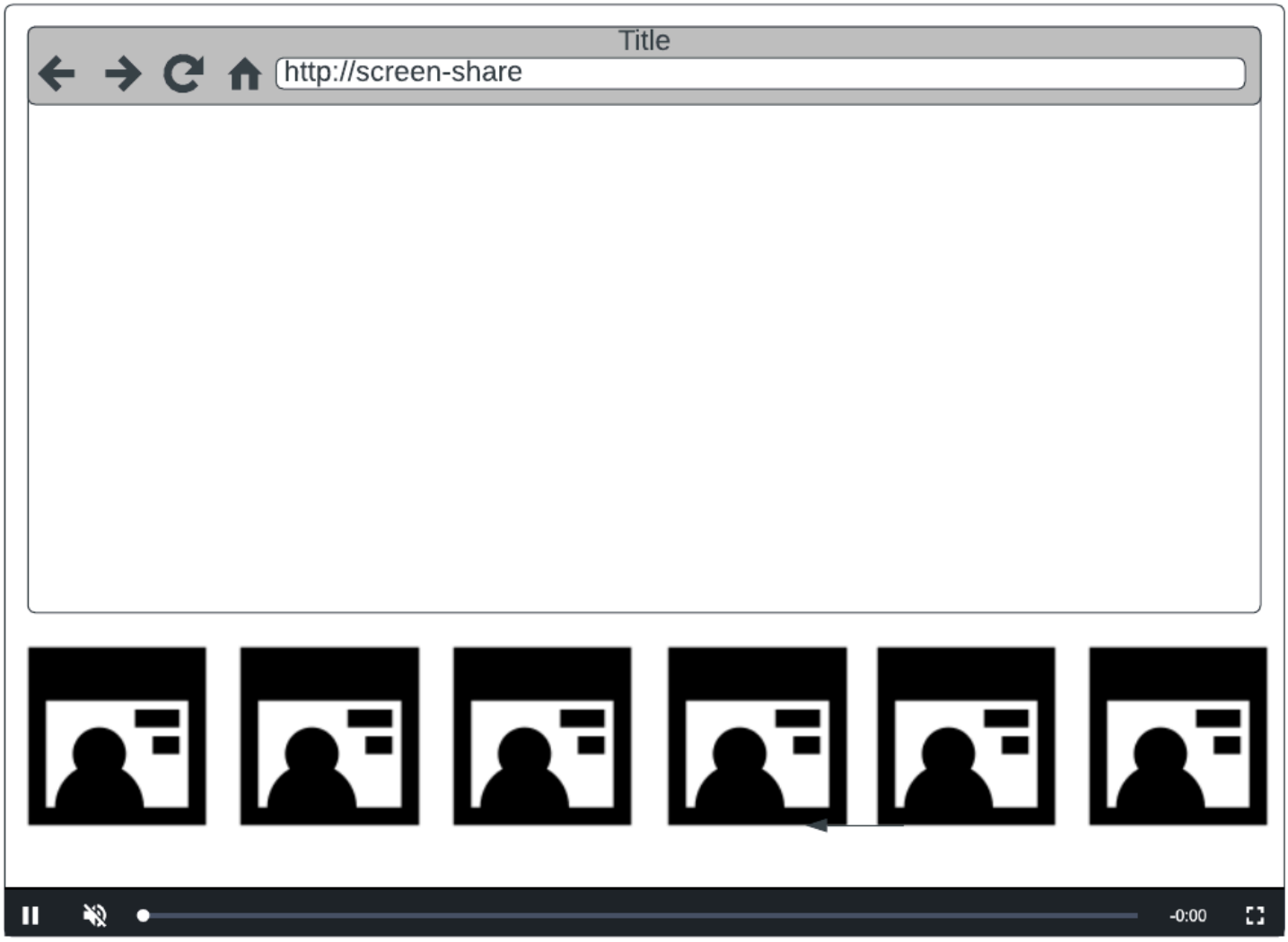


Per informazioni complete, consulta la Documentazione di riferimento delle API di streaming in tempo reale IVS.

Layouts (Layout)

Per impostazione predefinita, la funzionalità di composizione lato server utilizza un layout a griglia per disporre i partecipanti alla fase in slot di dimensioni uguali:

Questo layout offre ai clienti la possibilità di configurare e richiamare uno slot in evidenza. Lo slot in evidenza si trova nella schermata principale, mentre gli altri partecipanti sono mostrati sotto di esso in slot di dimensioni uguali:



Nota: la risoluzione massima supportata da un editore della fase per la composizione lato server è 1080p. Se un editore invia un video con risoluzione superiore a 1080p, verrà visualizzato come partecipante solo audio.

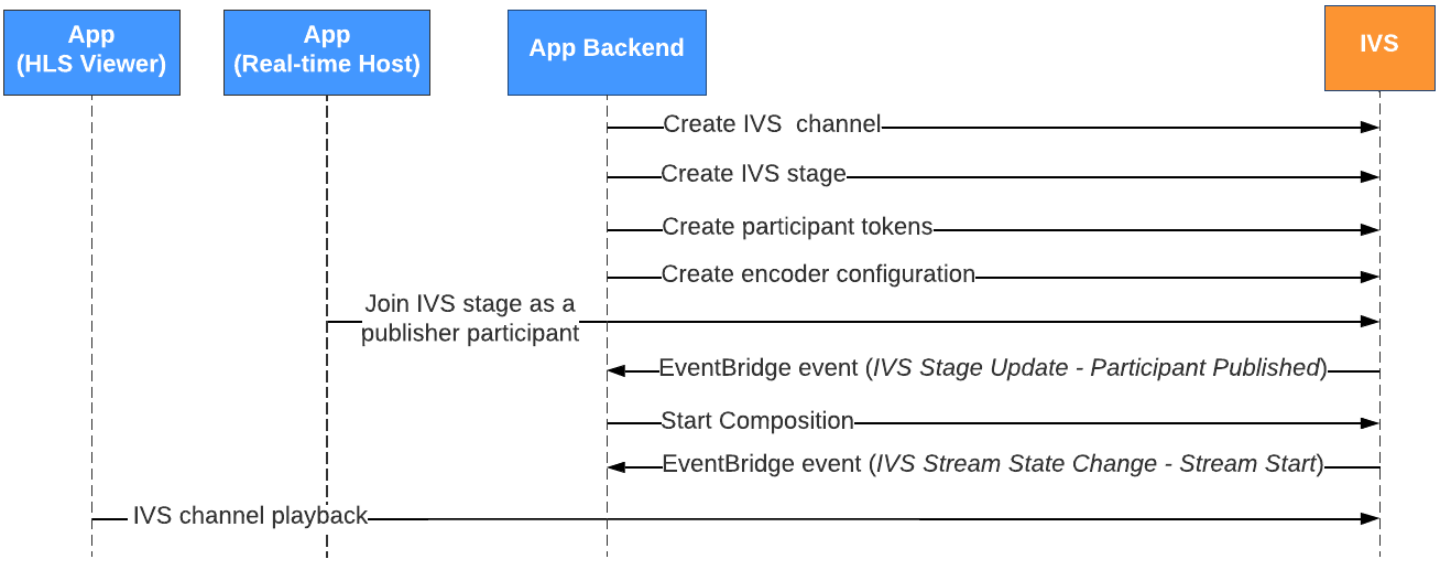
Nozioni di base

Prerequisiti

Per utilizzare la composizione lato server, è necessario disporre di una fase con editori attivi e utilizzare un canale IVS e/o un bucket S3 come destinazione della composizione. Di seguito, descriviamo un possibile flusso di lavoro che utilizza gli eventi EventBridge per avviare una composizione che trasmetta la fase su un canale IVS quando un partecipante pubblica. In alternativa, puoi avviare e interrompere le composizioni in base alla logica della tua applicazione. Consulta

Registrazione composita per un altro esempio che mostra l'uso della composizione lato server per registrare una fase direttamente in un bucket S3.

1. Crea un canale IVS. Consulta Guida introduttiva allo streaming a bassa latenza di Amazon IVS.
2. Crea una fase IVS e dei token per i partecipanti per ogni editore.
3. Crea un EncoderConfiguration.
4. Unisciti alla fase e pubblica su di essa. (Consulta le sezioni "Pubblicazione e sottoscrizione" delle guide per l'SDK di trasmissione in streaming in tempo reale: Web, Android e iOS).
5. Quando ricevi un evento EventBridge Partecipante pubblicato, chiama StartComposition.
6. Attendi qualche secondo e guarda la vista composita nella riproduzione del canale.



Nota: una composizione si spegne automaticamente dopo 60 secondi di inattività degli editori che partecipano alla fase. A quel punto, la composizione è terminata e passa a uno stato `STOPPED`. Una composizione viene eliminata automaticamente dopo alcuni minuti nello stato `STOPPED`.

Istruzioni per la CLI

L'uso di AWS CLI è un'opzione avanzata e richiede prima il download e la configurazione della CLI sul computer. Per maggiori dettagli, consultare la Guida per l'utente dell'interfaccia a riga di comando di AWS.

Ora puoi usare la CLI per creare e gestire le risorse. Gli endpoint della composizione si trovano nello spazio dei nomi `ivs-realtime`.

Creazione della risorsa EncoderConfiguration

Un oggetto EncoderConfiguration consente di personalizzare il formato del video da generare (altezza, larghezza, bitrate e altri parametri di streaming). È possibile riutilizzare un EncoderConfiguration ogni volta che si chiama l'endpoint di composizione, come spiegato nel passaggio successivo.

Il comando seguente crea una risorsa EncoderConfiguration che configura i parametri di composizione video lato server (bitrate video, framerate e risoluzione):

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

La risposta è:

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

Inizio di una composizione

Utilizzando l'ARN di EncoderConfiguration fornito nella risposta precedente, crea la tua risorsa di composizione:

```
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 risposta mostrerà che la composizione è stata creata con uno stato `STARTING`. Una volta che Composition inizia a pubblicare la composizione, lo stato passa a `ACTIVE`. (Puoi vedere lo stato chiamando l'endpoint ListCompositions o GetComposition.)

Una volta che la composizione è `ACTIVE`, la vista composita della fase IVS sarà visibile sul canale IVS, tramite ListCompositions:

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

La risposta è:

```
{
   "compositions": [
      {
         "arn": "arn:aws:ivs:us-east-1:927810967299:composition/YVoaXkKdEdRP",
         "destinations": [
            {
               "id": "bD9rRoN91fHU",
               "startTime": "2023-09-21T15:38:39+00:00",
               "state": "ACTIVE"
            }
         ],
         "stageArn": "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
         "startTime": "2023-09-21T15:38:37+00:00",
         "state": "ACTIVE",
         "tags": {}
      }
   ]
}
```

Nota: per mantenere viva la composizione, è necessario che i partecipanti editori pubblichino attivamente nella fase. Per ulteriori informazioni, consulta le sezioni "Pubblicazione e sottoscrizione" delle guide per l'SDK di trasmissione in streaming in tempo reale: Web, Android e iOS. È necessario creare un token di fase distinto per ogni partecipante.

Abilitazione della condivisione dello schermo

Per utilizzare un layout di condivisione dello schermo fisso, completa la procedura riportata di seguito.

Creazione della risorsa EncoderConfiguration

Il comando seguente crea una risorsa EncoderConfiguration che configura i parametri di composizione lato server (bitrate video, framerate e risoluzione).

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

Crea un token per i partecipanti alla fase con un attributo `screen-share`. Poiché specificheremo `screen-share` come nome dello slot `featured`, dobbiamo creare un token di fase con l'attributo `screen-share` impostato su `true`:

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

La risposta è:

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

Inizio della composizione

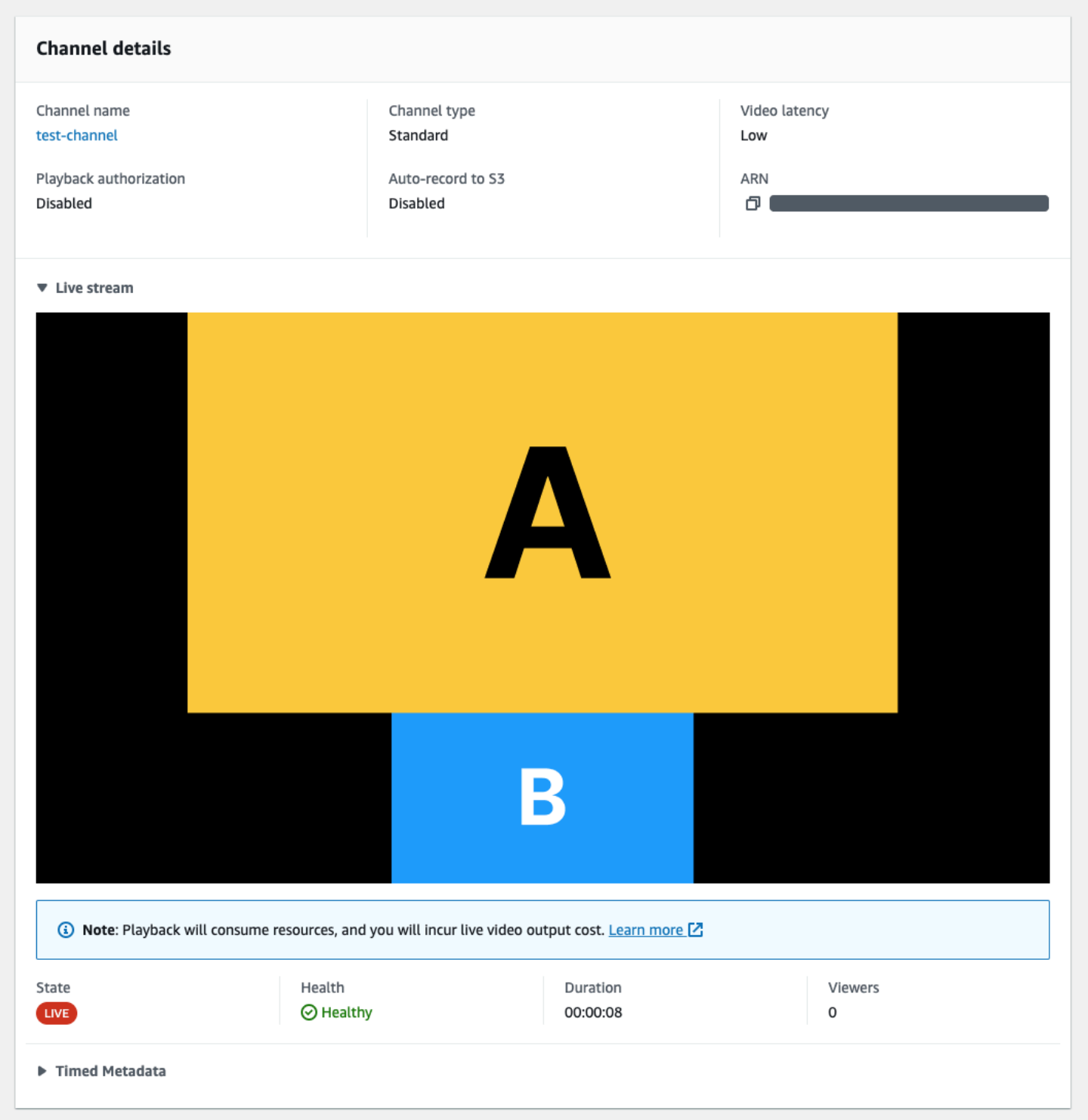
Per iniziare la composizione utilizzando la funzione di condivisione dello schermo, utilizziamo questo comando:

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVaOwO"}}]' --layout
 "grid={featuredParticipantAttribute=screen-share}"
```

La risposta è:

```
{
   "composition" : {
      "arn" : "arn:aws:ivs:us-east-1:927810967299:composition/B19tQcXRgtoz",
      "destinations" : [ {
         "configuration" : {
            "channel" : {
               "channelArn" : "arn:aws:ivs:us-east-1:927810967299:channel/
DOlMW4dfMR8r",
               "encoderConfigurationArn" : "arn:aws:ivs:us-east-1:927810967299:encoder-
configuration/DEkQHWPVaOwO"
            },
            "name" : ""
         },
         "id" : "SGmgBXTULuXv",
         "state" : "STARTING"
      } ],
      "layout" : {
         "grid" : {
            "featuredParticipantAttribute" : "screen-share"
         }
      },
      "stageArn" : "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
      "startTime" : "2023-09-27T21:32:38Z",
      "state" : "STARTING",
      "tags" : { }
   }
}
```

Quando il partecipante alla fase `E813MFklPWLF` si unisce alla fase, il video di quel partecipante verrà visualizzato nello slot in evidenza e tutti gli altri editori della fase saranno visualizzati sotto lo slot:



Arresto della composizione

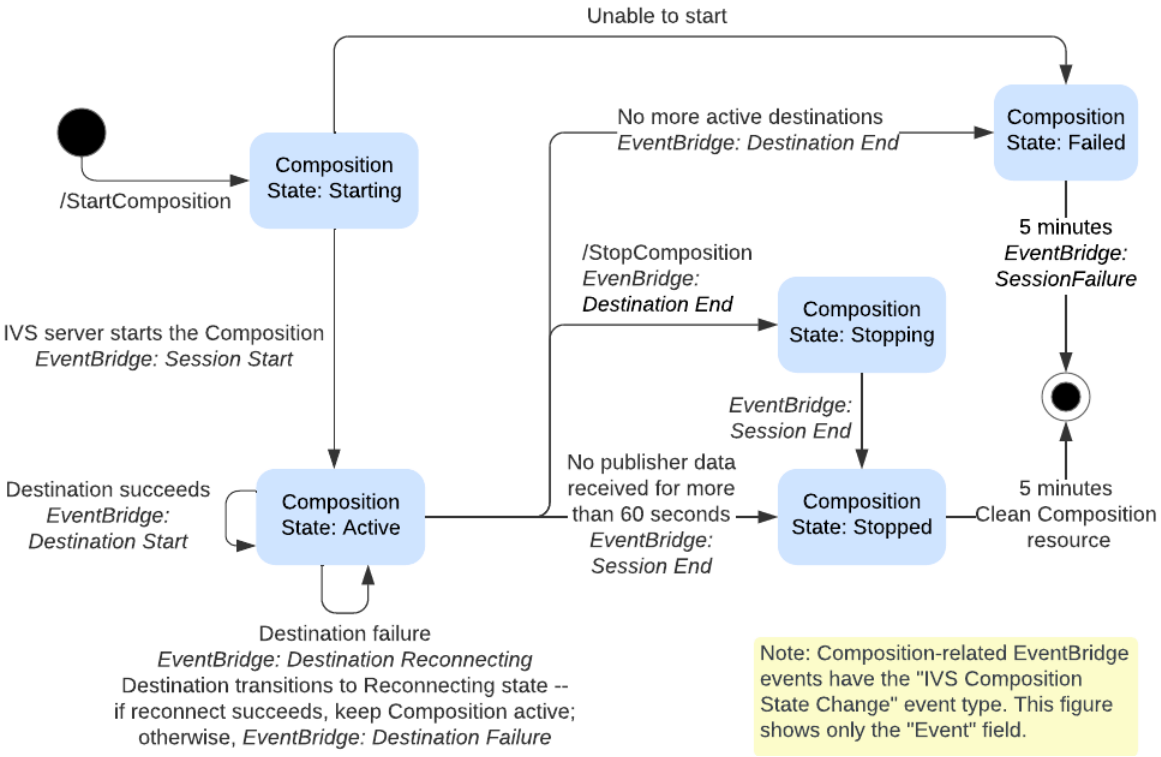
Per arrestare una composizione in qualsiasi momento, chiama l'endpoint StopComposition:

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

Ciclo di vita della composizione

Usa il diagramma seguente per comprendere le transizioni di stato di una composizione. In generale, il ciclo di vita di una composizione è il seguente:

1. Quando l'utente chiama l'endpoint StartComposition, viene creata una risorsa della composizione
2. Una volta che IVS avvia correttamente la composizione, viene inviato un evento EventBridge "Modifica allo stato della composizione IVS (Inizio sessione)". Consulta Utilizzo di EventBridge con lo streaming in tempo reale IVS per i dettagli sugli eventi.
3. Una volta che una composizione è in uno stato attivo, può verificarsi quanto segue:
   - L'utente interrompe la composizione: se viene chiamato l'endpoint StopComposition, IVS avvia un arresto graduale della composizione, inviando gli eventi "Fine destinazione" seguiti da un evento "Fine sessione".
   - La composizione si spegne automaticamente: se nessun partecipante pubblica attivamente nella fase IVS, la composizione viene finalizzata automaticamente dopo 60 secondi e vengono inviati gli eventi EventBridge.
   - Errore di destinazione: se una destinazione riporta inaspettatamente un errore (ad esempio, il canale IVS viene eliminato), la destinazione passa allo stato `RECONNECTING` e viene inviato un evento "Riconnessione della destinazione". Se il ripristino è impossibile, IVS trasferisce la destinazione allo stato `FAILED` e viene inviato un evento "Errore destinazione". IVS mantiene viva la composizione se almeno una delle sue destinazioni è attiva.
4. Una volta che la composizione è nello stato `STOPPED` o `FAILED`, viene ripulita automaticamente dopo cinque minuti. (Quindi non viene più recuperata da ListCompositions o 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.

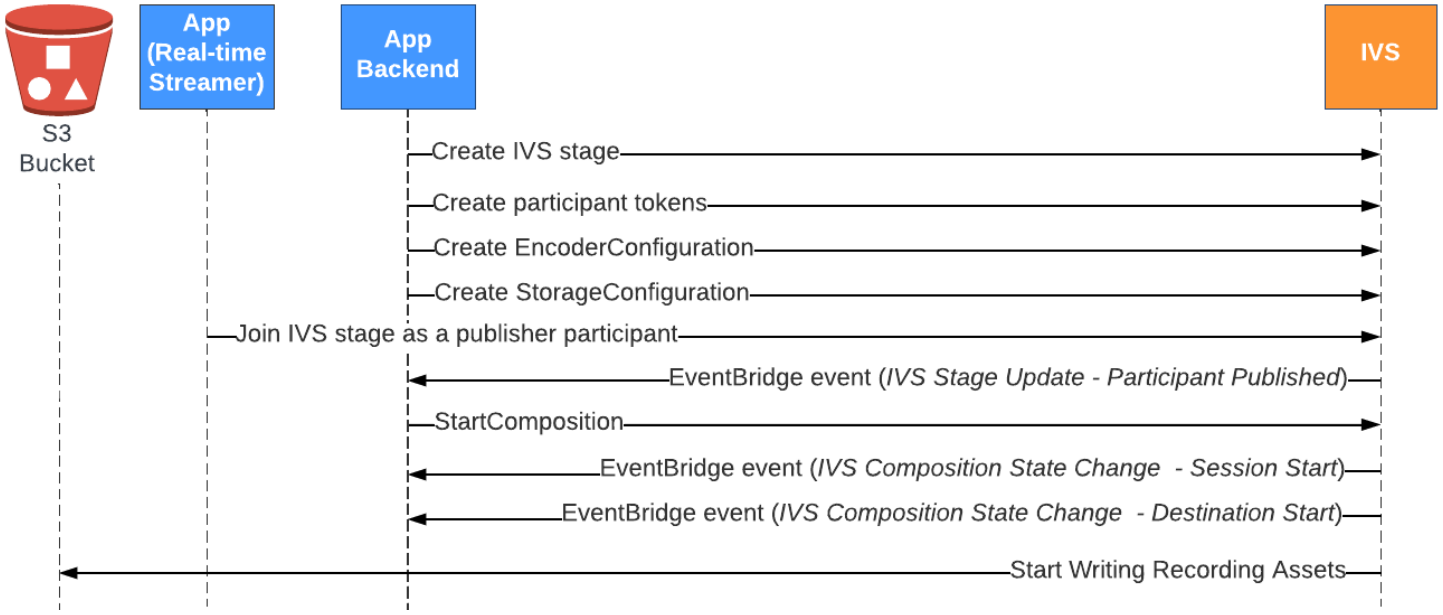
Registrazione composita (streaming in tempo reale)

Questo documento spiega come utilizzare la funzionalità di registrazione composita all'interno della composizione lato server. La registrazione composita consente di generare registrazioni HLS di una fase IVS combinando efficacemente tutti gli editori della fase in un'unica visualizzazione utilizzando un server IVS e quindi salvando il video risultante in un bucket S3.

Prerequisiti

Per utilizzare la registrazione composita, è necessario disporre di uno stage con editori attivi e di un bucket S3 da utilizzare come destinazione di registrazione. Di seguito, descriviamo un possibile flusso di lavoro che utilizza EventBridge gli eventi per registrare una composizione in un bucket S3. In alternativa, puoi avviare e interrompere le composizioni in base alla logica della tua applicazione.

1. Crea una fase IVS e dei token per i partecipanti per ogni editore.
2. Crea un EncoderConfiguration(un oggetto che rappresenta come deve essere renderizzato il video registrato).
3. Crea un bucket S3 e un StorageConfiguration(dove verranno archiviati i contenuti della registrazione).
4. Unisciti alla fase e pubblica su di essa.
5. Quando ricevi un EventBridge evento Participant Published, chiama StartCompositioncon un oggetto S3 DestinationConfiguration come destinazione
6. Dopo alcuni secondi, dovresti essere in grado di vedere i segmenti HLS mantenuti nei tuoi bucket S3.



Nota: una composizione si spegne automaticamente dopo 60 secondi di inattività degli editori che partecipano alla fase. A quel punto, la composizione viene terminata e passa a uno stato `STOPPED`. Una composizione viene eliminata automaticamente dopo alcuni minuti nello stato `STOPPED`. Per i dettagli, consulta Ciclo di vita della composizione in Composizione lato server.

Esempio di registrazione composita: StartComposition con una destinazione S3 Bucket

L'esempio seguente mostra una tipica chiamata all'StartCompositionendpoint, che specifica S3 come unica destinazione per la composizione. Una volta che la composizione passa a uno stato `ACTIVE`, i segmenti video e i metadati inizieranno a essere scritti nel bucket S3 specificato dall'oggetto `storageConfiguration`. Per creare composizioni con layout diversi, consulta "Layout" in Composizione lato server e la Documentazione di riferimento delle API dello streaming in tempo reale IVS.

Richiesta

```
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"
}
```

Risposta

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

Il `recordingPrefix` campo, presente nella StartComposition risposta, può essere utilizzato per determinare dove verranno archiviati i contenuti della registrazione.

Contenuto della registrazione

Quando la composizione passa a uno `ACTIVE` stato, inizierai a vedere i segmenti video HLS e i file di metadati che vengono scritti nel bucket S3 fornito durante la chiamata. StartComposition Questi contenuti sono disponibili per la post-elaborazione o la riproduzione come video on demand.

Tieni presente che dopo che una composizione diventa attiva, viene emesso un evento "Cambia stato della composizione IVS" ed è possibile che passi del tempo prima che i file manifesto e i segmenti video vengano scritti. Consigliamo di riprodurre o elaborare flussi registrati solo dopo che è stato ricevuto l'evento "Modifica dello stato di composizione dell'IVS (Fine sessione)". Per i dettagli, consulta Utilizzo con IVS Real-Time Streaming. EventBridge

Di seguito è riportato un esempio di contenuti e della struttura di directory di una registrazione di una sessione IVS live:

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

La cartella `events` contiene i file di metadati corrispondenti all'evento di registrazione. I file di metadati JSON vengono generati quando la registrazione inizia, termina correttamente o termina con errori:

- `events/recording-started.json`
- `events/recording-ended.json`

- `events/recording-failed.json`

Una determinata cartella `events` conterrà `recording-started.json` e `recording-ended.json` o `recording-failed.json`.

Questi contengono metadati relativi alla sessione registrata e ai relativi formati di output. I dettagli JSON sono riportati di seguito.

La cartella `media` contiene i contenuti multimediali supportati. La sottocartella `hls` contiene tutti i file multimediali e i file manifesto generati durante la sessione della composizione ed è riproducibile con il lettore IVS. Il manifesto HLS si trova nella cartella `multivariant.m3u8`.

Bucket Policy per StorageConfiguration

Quando viene creato un StorageConfiguration oggetto, IVS avrà accesso alla scrittura di contenuti nel bucket S3 specificato. Questo accesso viene concesso apportando modifiche alla policy del bucket S3. Se la policy per il bucket viene modificata in modo da rimuovere l'accesso di IVS, le registrazioni in corso e quelle nuove avranno esito negativo.

L'esempio seguente mostra una policy di bucket S3 che consente a IVS di scrivere nel bucket 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"
```

```
                }
            }
        }
    ]
}
```

File di metadati JSON

Questi metadati sono in formato JSON. Tale controllo contiene le seguenti informazioni:

Campo	Tipo	Campo obbligatorio	Descrizione
`stage_arn`	stringa	Sì	ARN della fase utilizzato come origine della composizione.
`media`	oggetto	Sì	L'oggetto che contiene gli oggetti enumerati del contenuto multimediale disponibile per la registrazione. Valori validi: `"hls"`.
`hls`	oggetto	Sì	Il campo enumerato che descrive l'output in formato Apple HLS.
`duration_ms`	intero	Condizionale	La durata del contenuto HLS registrato, in millisecondi. Questo valore è disponibile solo quando `recording_status` è `"RECORDING_ENDED"` o`"RECORDING_ENDED_WITH_FAILURE"` . Se prima di una registrazione si è verificato un errore, allora sarà uguale a 0.
`path`	stringa	Sì	Il percorso relativo dal prefisso S3 in cui è memorizzato il contenuto HLS.

Campo	Tipo	Campo obbligatorio	Descrizione
`playlist`	stringa	Sì	Il nome del file della playlist principale HLS.
`renditions`	oggetto	Sì	L'array di rendering (variante HLS) di oggetti di metadati. È presente sempre almeno un rendering.
`path`	stringa	Sì	Il percorso relativo dal prefisso S3 in cui è memorizzato il contenuto HLS per questo rendering.
`playlist`	stringa	Sì	Il nome del file della playlist multimediale per questo rendering.
`resolution_height`	int	Condizionale	L'altezza della risoluzione in pixel del video codificato. Questa opzione è disponibile solo quando il rendering contiene una traccia video.
`resolution_width`	int	Condizionale	La larghezza della risoluzione in pixel del video codificato. Questa opzione è disponibile solo quando il rendering contiene una traccia video.

Campo	Tipo	Campo obbligatorio	Descrizione
`recording_ended_at`	string	Condizionale	Il timestamp UTC di RFC 3339 quando la registrazione termina. Questo valore è disponibile solo quando `recording_status` è `"RECORDING_ENDED"` o `"RECORDING_ENDED_WITH_FAILURE"` . `recording_started_at` e `recording_ended_at` sono timestamp quando questi eventi vengono generati e potrebbero non corrispondere esattamente ai timestamp del segmento video HLS. Per determinare con precisione la durata di una registrazione, utilizzare il campo `duration_ms` .
`recording_started_at`	string	Condizionale	Il timestamp UTC di RFC 3339 quando la registrazione inizia. Questo non è disponibile quando `recording_status` è `RECORDING_START_FAILED` . Consultare la nota sopra per `recording_ended_at` .

Campo	Tipo	Campo obbligatorio	Descrizione
`recording_status`	string	Sì	Lo stato della registrazione. Valori validi: `"RECORDING_STARTED"`, `"RECORDING_ENDED"`, `"RECORDING_START_FAILED"`, `"RECORDING_ENDED_WITH_FAILURE"`.
`recording_status_message`	string	Condizionale	Le informazioni descrittive sullo stato. Questo valore è disponibile solo quando `recording_status` è `"RECORDING_ENDED"` o `"RECORDING_ENDED_WITH_FAILURE"`.
`version`	string	Sì	La versione dello schema dei metadati.

Esempio: 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
        }
```

```
      ]
    }
  }
}
```

Esempio: 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
        }
      ]
    }
  }
}
```

Esempio: 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
        }
      ]
    }
  }
}
```

Riproduzione di contenuti registrati da bucket privati

Per impostazione predefinita, il contenuto registrato è privato, pertanto questi oggetti sono inaccessibili per la riproduzione direttamente tramite l'URL S3. Se provi ad aprire la playlist multivariata HLS (file m3u8) per la riproduzione utilizzando il lettore IVS o un altro lettore, riceverai un errore (ad esempio, "Non disponi dell'autorizzazione per accedere alla risorsa richiesta"). Puoi invece riprodurre questi file con Amazon CloudFront CDN (Content Delivery Network).

CloudFront le distribuzioni possono essere configurate per fornire contenuti provenienti da bucket privati. In genere è preferibile disporre di bucket apertamente accessibili in cui le letture aggirano i controlli offerti da. CloudFront È possibile configurare la distribuzione in modo che venga servita da un bucket privato creando un Origin Access Control (OAC), ovvero un CloudFront utente speciale con autorizzazioni di lettura sul bucket di origine privato. Puoi creare l'OAC dopo aver creato la tua distribuzione, tramite la console o l'API. CloudFront Vedi Creazione di un nuovo controllo di accesso all'origine nella Amazon CloudFront Developer Guide.

Configurazione della riproduzione utilizzando CORS CloudFront Enabled

Questo esempio illustra come uno sviluppatore può configurare una CloudFront distribuzione con CORS abilitato, abilitando la riproduzione delle proprie registrazioni da qualsiasi dominio. Ciò è particolarmente utile durante la fase di sviluppo, ma è possibile modificare l'esempio seguente per adattarlo alle proprie esigenze di produzione.

Fase 1: creazione di un bucket S3

Crea un bucket S3 che verrà utilizzato per archiviare le registrazioni. Tieni presente che il bucket dovrà trovarsi nella stessa Regione utilizzata per il flusso di lavoro IVS.

Aggiungi una policy CORS di autorizzazione al bucket:

1. Nella console AWS, passa alla scheda Autorizzazioni del bucket S3.
2. Copia la policy CORS riportata di seguito e incollala in Cross-Origin Resource Sharing (CORS). Ciò consentirà l'accesso CORS al bucket S3.

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

Fase 2: Creare una distribuzione CloudFront

Vedi Creazione di una CloudFront distribuzione nella Guida per CloudFront gli sviluppatori.

Dalla console AWS, immetti le seguenti informazioni aziendali:

Per questo campo…	Scegli questo...
Dominio origine	Il bucket S3 creato nella fase precedente
Accesso all'origine	Impostazioni di controllo dell'accesso all'origine (consigliato), utilizzando parametri predefiniti

Per questo campo…	Scegli questo...
Comportamento predefinito della cache: policy del protocollo per i visualizzatori	Reindirizza HTTP a HTTPS
Comportamento predefinito della cache: Metodi HTTP consentiti	GET, HEAD e OPTIONS
Comportamento predefinito della cache: chiavi di cache e richieste di origine	CachingDisabled politica
Comportamento predefinito della cache: policy di richiesta dell'origine	CORS-S3Origin
Comportamento predefinito della cache: policy delle intestazioni di risposta	SimpleCORS
Web Application Firewall	Abilitazione delle protezioni di sicurezza

Quindi salva la CloudFront distribuzione.

Fase 3: Configurazione della policy del bucket S3

1. Elimina tutto StorageConfiguration ciò che hai configurato per il bucket S3. Ciò rimuoverà tutte le policy del bucket che sono state aggiunte automaticamente durante la creazione della policy per quel bucket.
2. Vai alla tua sezione CloudFront Distribution, assicurati che tutti i campi di distribuzione siano negli stati definiti nel passaggio precedente e Copia la Bucket Policy (usa il pulsante Copia policy).
3. Passa al bucket S3. Nella scheda Autorizzazioni, seleziona Modifica la policy del bucket e incolla la policy del bucket copiata nel passaggio precedente. Dopo questo passaggio, la bucket policy dovrebbe includere esclusivamente la CloudFront policy.
4. Crea un StorageConfiguration, specificando il bucket S3.

Dopo StorageConfiguration la creazione, vedrai due elementi nella policy del bucket S3, uno che consente di leggere i contenuti e l'altro che consente CloudFront a IVS di scrivere contenuti. Un esempio di policy bucket finale, con accesso IVS, è mostrato in Esempio: S3 Bucket Policy with CloudFront and IVS Access. CloudFront

Fase 4: Riproduzione delle registrazioni

Dopo aver impostato correttamente la CloudFront distribuzione e aggiornato la policy sui bucket, dovresti essere in grado di riprodurre le registrazioni utilizzando il lettore IVS:

1. Avvia correttamente una composizione e assicurati di avere una registrazione memorizzata nel bucket S3.
2. Dopo aver seguito i passaggi da 1 a 3 in questo esempio, i file video dovrebbero essere disponibili per l'utilizzo tramite l'URL. CloudFront Il tuo CloudFront URL è il nome del dominio di distribuzione nella scheda Dettagli della CloudFront console Amazon. Dovrebbe essere simile a quanto segue:

   ```
   a1b23cdef4ghij.cloudfront.net
   ```
3. Per riprodurre il video registrato tramite la CloudFront distribuzione, trova la chiave oggetto del `multivariant.m3u8` file nel bucket s3. Dovrebbe essere simile a quanto segue:

   ```
   FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/
   multivariant.m3u8
   ```
4. Aggiungi la chiave dell'oggetto alla fine dell'URL. CloudFront Il proprio URL finale sarà simile a quanto segue:

   ```
   https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/
   fjFKbylPb3k4/composite/media/hls/multivariant.m3u8
   ```
5. Ora puoi aggiungere l'URL finale all'attributo di origine di un lettore IVS per guardare la registrazione completa. Per guardare il video registrato, puoi utilizzare la demo in Guida introduttiva nell'SDK del lettore IVS: Guida per il Web.

Esempio: S3 Bucket Policy con CloudFront accesso IVS

Lo snippet riportato di seguito illustra una policy sui bucket S3 che consente CloudFront di leggere i contenuti nel bucket privato e IVS di scrivere contenuti nel bucket. Nota: non copiare e incollare il frammento di codice riportato di seguito nel tuo bucket. La tua policy dovrebbe contenere gli ID pertinenti alla tua distribuzione e. CloudFront 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"
        }
      }
    }
  ]
}
```

Risoluzione dei problemi

- La composizione non è scritta nel bucket S3: assicurati che il bucket S3 e StorageConfiguration gli oggetti siano creati e si trovino nella stessa regione. Assicurati inoltre che IVS abbia accesso al bucket controllando la tua policy sul bucket; consulta Bucket Policy per. StorageConfiguration

- Non riesco a trovare una composizione durante l'esecuzione ListCompositions: le composizioni sono risorse effimere. Una volta passate allo stato finale, vengono eliminate automaticamente dopo alcuni minuti.
- La mia composizione si interrompe automaticamente: una composizione si interrompe automaticamente se non c'è nessun editore nella fase per più di 60 secondi.

Problema noto

La playlist multimediale scritta mediante registrazione composita ha il tag `#EXT-X-PLAYLIST-TYPE:EVENT` mentre la composizione è in corso. Al termine della composizione, il tag viene aggiornato a `#EXT-X-PLAYLIST-TYPE:VOD`. Per un'esperienza di riproduzione fluida, ti consigliamo di utilizzare questa playlist solo dopo che la composizione è stata completata con successo.

Supporto OBS e WHIP (streaming in tempo reale)

Questo documento spiega come utilizzare codificatori compatibili con WHIP come OBS per pubblicare su IVS lo streaming in tempo reale. WHIP (WebRTC-HTTP Ingestion Protocol) è una bozza IETF sviluppata per standardizzare l'ingestione di WebRTC.

WHIP consente la compatibilità con software come OBS, offrendo un'alternativa (all'IVS broadcast SDK) per il desktop publishing. Gli streamer più sofisticati che hanno familiarità con OBS potrebbero preferirlo per le sue funzionalità di produzione avanzate, come le transizioni di scena, il mixaggio audio e la grafica di sovrapposizione. Ciò offre agli sviluppatori un'opzione versatile: utilizzate l'IVS Web Broadcast SDK per la pubblicazione diretta su browser o consentite agli streamer di utilizzare OBS sul proprio desktop per ottenere strumenti più potenti.

Inoltre, WHIP è utile in situazioni in cui l'utilizzo dell'SDK di trasmissione IVS non è fattibile o preferibile. Ad esempio, nelle configurazioni che coinvolgono codificatori hardware, l'SDK di trasmissione IVS potrebbe non essere un'opzione. Tuttavia, se l'encoder supporta WHIP, è comunque possibile pubblicare direttamente dall'encoder su IVS.

Guida OBS

OBS supporta WHIP a partire dalla versione 30. Per iniziare, scarica OBS v30 o versione successiva: https://obsproject.com/.

Per pubblicare su uno stage IVS utilizzando OBS tramite WHIP, segui questi passaggi:

1. Genera un token partecipante con funzionalità di pubblicazione. In termini WHIP, un token partecipante è un token al portatore. Per impostazione predefinita, i token dei partecipanti scadono dopo 12 ore, ma è possibile estenderne la durata fino a 14 giorni.
2. Fare clic su Settings (Impostazioni). Nella sezione Stream del pannello Impostazioni, seleziona WHIP dal menu a discesa Servizio.
3. Per il Server, inserisci https://global.whip.live-video.net/.
4. Per il token Bearer, inserisci il token partecipante che hai generato nel passaggio 2.
5. Configura le impostazioni video come faresti normalmente, con alcune restrizioni:
   a. Lo streaming in tempo reale IVS supporta input fino a 720p a 8,5 Mbps. Se superi uno di questi limiti, lo streaming verrà disconnesso.

b. Ti consigliamo di impostare l'intervallo dei fotogrammi chiave nel pannello Output su 1 o 2 secondi. Un intervallo di fotogrammi chiave basso consente agli spettatori di iniziare più rapidamente la riproduzione dei video. Consigliamo inoltre di impostare CPU Usage Preset su ultraveloce e Tune su latenza zero, per abilitare la latenza più bassa.

c. Poiché OBS non supporta il simulcast, consigliamo di mantenere il bitrate al di sotto di 2,5 Mbps. Ciò consente agli spettatori con connessioni a larghezza di banda inferiore di guardare.

6. Premi Avvia streaming.

Service Quotas (streaming in tempo reale)

Di seguito sono riportati i limiti e le service quotas per gli endpoint, le risorse e altre operazioni in tempo reale di Amazon Interactive Video Service (IVS). Le service quotas (quote di servizio), a cui si fa riferimento anche come limiti, rappresentano il numero massimo di risorse di servizio o operazioni per l'account AWS. In altre parole, questi limiti sono per account AWS se non diversamente indicato nella tabella. Consulta anche Service Quotas AWS.

Per connetterti a livello di programmazione a un servizio AWS, utilizza un endpoint. Consulta anche Endpoint di servizio AWS.

Tutte le quote vengono applicate per regione.

Aumento delle quote di servizio

Per le quote regolabili, è possibile richiedere un aumento tramite la console AWS. Utilizzare la console per visualizzare informazioni anche sulle service quotas (quote di servizio).

Le quote tariffarie delle chiamate API non sono regolabili.

Quote tariffarie per le chiamate API

Tipo di endpoint	Endpoint	Predefinita
Composizione	`GetComposition`	5 TPS
Composizione	`ListCompositions`	5 TPS
Composizione	`StartComposition`	5 TPS
Composizione	`StopComposition`	5 TPS
MediaEncoder	`CreateEncoderConfiguration`	5 TPS
MediaEncoder	`DeleteEncoderConfiguration`	5 TPS
MediaEncoder	`GetEncoderConfiguration`	5 TPS
MediaEncoder	`ListEncoderConfigurations`	5 TPS

Tipo di endpoint	Endpoint	Predefinita
Stage	`CreateParticipantToken`	50 TPS
Stage	`CreateStage`	5 TPS
Stage	`DeleteStage`	5 TPS
Stage	`DisconnectParticipant`	5 TPS
Stage	`GetParticipant`	5 TPS
Stage	`GetStage`	5 TPS
Stage	`GetStageSession`	5 TPS
Stage	`ListStages`	5 TPS
Stage	`UpdateStage`	5 TPS
Stage	`ListParticipants`	5 TPS
Stage	`ListParticipantEvents`	5 TPS
Stage	`ListStageSessions`	5 TPS
StorageConfiguration	`CreateStorageConfiguration`	5 TPS
StorageConfiguration	`DeleteStorageConfiguration`	5 TPS
StorageConfiguration	`GetStorageConfiguration`	5 TPS
StorageConfiguration	`ListStorageConfigurations`	5 TPS
Tag	`ListTagsForResource`	10 TPS
Tag	`TagResource`	10 TPS
Tag	`UntagResource`	10 TPS

Other Quotas (Altre quote)

Risorsa o funzionalità	Predefinita	Adattabile	Description
EncoderConfigurations	20	Sì	Numero massimo di risorse di configurazione dell'encoder per account.
Destinazioni di composizione	2	No	Numero massimo di oggetti di destinazione in una risorsa di composizione.
Composizione: durata massima	24	No	Tempo massimo di esistenza di una composizione, in ore.
Composizioni	5	Sì	Numero massimo di risorse di composizione simultanee per account.
Durata della pubblicazione o della sottoscrizione del partecipante	24	No	Periodo di tempo massimo in cui un partecipante può pubblicare o rimanere abbonato a uno stage, in ore.
Risoluzione di pubblicazione dei partecipanti	720p	No	Risoluzione massima del video pubblicato dai partecipanti.
Bitrate di download dei partecipanti	8,5 Mbps	No	Bitrate massimo di download aggregato per tutti gli abbonamenti di un partecipante.
Partecipanti alla fase (editori)	12	No	Numero massimo di partecipanti che possono pubblicare in una fase contemporaneamente.
Partecipanti alla fase (abbonati)	10.000	Sì	Numero massimo di partecipanti che possono abbonarsi

Risorsa o funzionalità	Predefinita	Adattabile	Description
			a una fase contempor aneamente.
Stage	100	Sì	Numero massimo di fasi, per regione AWS.

Ottimizzazioni dello streaming in tempo reale

Per garantire agli utenti la migliore esperienza di streaming e visualizzazione di video tramite lo streaming in tempo reale IVS, esistono diversi modi per migliorare o ottimizzare alcune parti dell'esperienza grazie alle funzionalità da noi offerte.

Introduzione

Quando si ottimizza per la qualità dell'esperienza di un utente, è importante considerare l'esperienza desiderata, che può cambiare a seconda dei contenuti che l'utente sta guardando e delle condizioni della rete.

In questa guida ci concentriamo sugli utenti che sono:editori di flussi o abbonati a flussi e consideriamo le operazioni e le esperienze desiderate di tali utenti.

Streaming adattivo: codifica a livelli con simulcast

Questa funzionalità è supportata solo nelle seguenti versioni client:

- iOS e Android 1.12.0 e versioni successive
- Web 1.5.1 e versioni successive

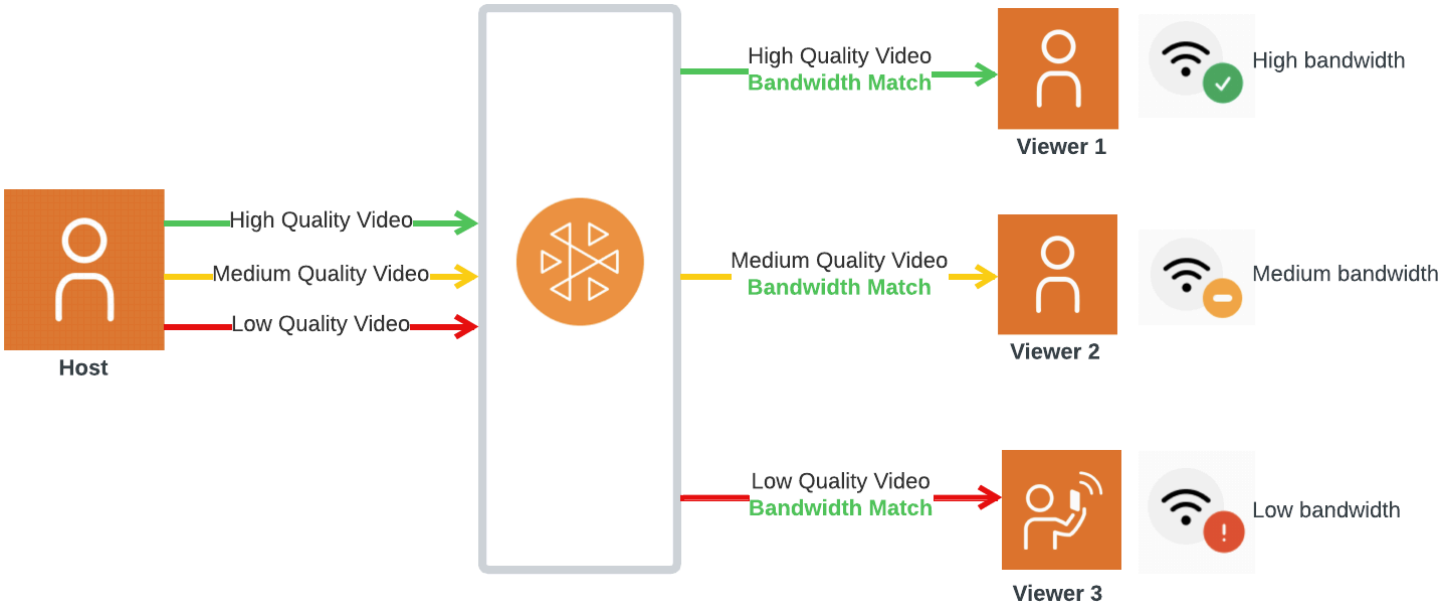
Devi inviare un'e-mail a amazon-ivs-simulcast@amazon .com per attivare questa funzionalità per il tuo account. L'abilitazione del simulcast tramite la configurazione SDK non avrà alcun effetto a meno che tu non abbia scelto di aderire.

Una volta scelta la funzionalità, quando si utilizzano gli SDK di trasmissione in tempo reale IVS, gli editori codificano più livelli di video e gli abbonati si adattano o passano automaticamente alla qualità più adatta alla loro rete. Questa è detta codifica a livelli con simulcast.

La codifica a livelli con simulcast è supportata su Android e iOS e sui browser desktop Chrome (per Windows e macOS). Non supportiamo la codifica a livelli su altri browser.

Nel diagramma seguente, l'host invia tre qualità video (alta, media e bassa). IVS trasmette il video con qualità più alta a ogni spettatore in base alla larghezza di banda disponibile; ciò fornisce un'esperienza ottimale per ogni spettatore. Se la connessione di rete dello spettatore 1 passa da

buona a cattiva, IVS inizia automaticamente a inviare un video di qualità inferiore in modo che spettatore 1 possa continuare a guardare lo streaming senza interruzioni (con la migliore qualità possibile).



Livelli, qualità e framerate predefiniti

Le qualità e i livelli predefiniti forniti per gli utenti mobili e Web sono i seguenti:

Cellulare (Android, iOS)	Web (Chrome)
Livello alto (o personalizzato): • Bitrate massimo: 900.000 bps • Framerate: 15 fps • Risoluzione: 360x640	Livello alto (o personalizzato): • Bitrate massimo: 1.700.000 bps • Framerate: 30 fps • Risoluzione: 1280x720
Livello intermedio: nessuno (non necessario poiché la differenza tra i bitrate di livello alto e basso su dispositivi mobili è ridotta)	Livello intermedio: • Bitrate massimo: 700.000 bps • Framerate: 20 fps • Risoluzione: 640x360
Livello basso: • Bitrate massimo: 150.000 bps	Livello basso: • Bitrate massimo: 200.000 bps

Cellulare (Android, iOS)	Web (Chrome)
• Framerate: 15 fps • Risoluzione: 180x320	• Framerate: 15 fps • Risoluzione: 320x180

Configurazione della codifica a livelli con simulcast

Per utilizzare la codifica a più livelli con simulcast, devi aver attivato la funzionalità e averla abilitata sul client. Se la abiliti, noterai un aumento del bitrate complessivo trasmesso, con il vantaggio di un minore congelamento dei video.

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
```

App

```
// Opt-out of Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})

// Other Stage implementation code
```

Configurazioni dello streaming

In questa sezione sono descritte altre configurazioni che possono essere impostate per i flussi audio e video.

Modifica del bitrate del flusso

Per modificare il bitrate del flusso video, usa i seguenti esempi di configurazione.

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
```

App

```
// 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
```

Modifica del framerate del flusso video

Per modificare il framerate del flusso video, usa i seguenti esempi di configurazione.

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
```

App

```
// 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
```

Ottimizzazione del bitrate audio e del supporto stereo

Per modificare le impostazioni bitrate e stereo del flusso audio, usa i seguenti esempi di configurazione.

App

```
// 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
```

Ottimizzazioni suggerite

Scenario	Raccomandazioni
Streaming con testo o contenuti a movimento lento, come presentazioni o diapositive	Usa la codifica a livelli con simulcast oppure configura i flussi con un framerate inferiore.
Streaming con azione o molto movimento	Usa la codifica a livelli con simulcast.
Streaming con conversazione o poco movimento	Usa la codifica a livelli con simulcast o scegli solo audio (consulta "Iscrizione ai partecipanti" nelle guide per gli SDK di trasmissione dello streaming in tempo reale: Web, Android e iOS).
Utenti che trasmettono in streaming con dati limitati	Utilizza la codifica a livelli con simulcast o, se desideri ridurre l'utilizzo dei dati per tutti, configura un framerate inferiore e riduci il bitrate manualmente.

Risorse e supporto (streaming in tempo reale)

Risorse

https://ivs.rocks/ è il sito dedicato per sfogliare contenuti pubblicati (demo, esempi di codice, post di blog), stimare i costi e sperimentare Amazon IVS attraverso demo live.

Demo

La demo di streaming in tempo reale di IVS per iOS e Android mostra agli sviluppatori come utilizzare Amazon IVS per creare un'interessante applicazione di contenuti in tempo reale generata dagli utenti social. Questa applicazione presenta un feed scorrevole di flussi in tempo reale generati dagli utenti. Gli utenti possono creare flussi video e stanze solo audio. Gli ospiti del flusso video possono partecipare in modalità guest spot o versus (VS). Le istruzioni su come implementare il back-end richiesto e creare l'applicazione sono disponibili nei seguenti repository GitHub:

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

- Android: https://github.com/aws-samples/amazon-ivs-real-time-for-android-demo/
- Back-end: https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

Supporto

Il Centro di supporto AWS offre un'ampia gamma di piani che forniscono l'accesso agli strumenti e alle competenze per supportare le soluzioni AWS. Tutti i piani di supporto forniscono accesso 24 ore su 24, 7 giorni su 7, al servizio clienti. Se ti occorrono supporto tecnico e risorse aggiuntive che aiutino a pianificare, implementare e ottimizzare l'ambiente AWS, puoi scegliere il piano di supporto che più si allinea al tuo caso d'uso AWS.

Supporto Premium AWS è un canale di supporto personale a risposta rapida per la creazione e l'esecuzione di applicazioni su AWS.

AWS re:Post è un sito di domande frequenti basato sulla community di sviluppatori e relativo a domande tecniche correlate ad Amazon IVS.

Contattaci: contiene collegamenti per domande generiche su fatturazione o account. Per quesiti tecnici, utilizzare i forum di discussione o i collegamenti di supporto elencati sopra.

Glossario

Consulta anche il glossario AWS. Nella tabella seguente, LL sta per IVS streaming a bassa latenza; RT per IVS streaming in tempo reale.

Termine	Descrizione	LL	RT	Chat
AAC	Codifica audio avanzata. AAC è uno standard di codifica audio per la compressione audio digitale con perdita. Progettato per essere il successore del formato MP3, AAC generalmente raggiunge una qualità del suono superiore rispetto all'MP3 a parità di bitrate. AAC è stato standardizzato da ISO e IEC come parte delle specifiche MPEG-2 e MPEG-4.	✓	✓	
Streaming con bitrate adattivo	Lo streaming con bitrate adattivo (ABR) consente al lettore IVS di passare a un bitrate inferiore quando la qualità della connessione ne risente e tornare a un bitrate più elevato quando la qualità migliora.	✓		
Streaming adattivo	Consulta la codifica a livelli con simulcast.		✓	
Utente amministratore	Un utente AWS con accesso amministrativo a risorse e servizi disponibili in un account AWS. Consulta la Terminologia nella Guida per l'utente alla configurazione di AWS.	✓	✓	✓
ARN	Nome della risorsa Amazon, identifica in modo univoco una risorsa AWS. I formati specifici ARN dipendono dal tipo di risorsa. Per i formati ARN utilizzati dalle risorse IVS, vedere Guida di riferimento sull'autorizzazione del servizio.	✓	✓	✓
Proporzioni	Descrive le proporzioni tra larghezza e altezza del frame. Ad esempio, 16:9 è la proporzione corrispondente alla risoluzione Full HD o 1080p.	✓	✓	

Termine	Descrizione	LL	RT	Chat
Modalità audio	Una configurazione audio preimpostata o personalizzata ottimizzata per diversi tipi di utenti di dispositivi mobili e per le apparecchiature utilizzate. Consulta SDK di trasmissione IVS: modalità audio per dispositivi mobili (streaming in tempo reale).		✓	
AVC, H.264, MPEG-4 Parte 10	Codifica video avanzata, nota anche come H.264 o MPEG-4 Parte 10, uno standard di compressione video per la compressione video digitale con perdita.	✓	✓	
Sostituzione dello sfondo	Un tipo di filtro della fotocamera che consente ai creatori di streaming live di modificare lo sfondo. Consulta Sostituzione dello sfondo in SDK di trasmissione IVS: filtri di fotocamere di terze parti (streaming in tempo reale).		✓	
Bitrate	Un parametro di streaming per il numero di bit trasmessi o ricevuti al secondo.	✓	✓	
Trasmissione, emittente	Altri termini per stream, streamer.	✓		
Buffering	Una condizione che si verifica quando il dispositivo di riproduzione non è in grado di scaricare il contenuto prima del momento in cui dovrebbe iniziare la riproduzione. Il buffering può manifestarsi in vari modi: il contenuto può interrompersi e riavviarsi in modo casuale (c.d. "stuttering"), il contenuto può interrompersi per lunghi periodi di tempo (c.d. "freezing") o il player IVS può sospendere la riproduzione.	✓	✓	

Termine	Descrizione	LL	RT	Chat
Playlist con intervalli di byte	Una playlist più granulare rispetto alla playlist HLS standard. La playlist HLS standard è composta da file multimediali di 10 secondi. Con una playlist con intervallo di byte, la durata del segmento è la stessa dell'intervallo di fotogrammi chiave configurato per lo streaming. La playlist con intervallo di byte è disponibile solo per le trasmissioni registrate automaticamente su un bucket S3. Viene creata in aggiunta alla playlist HLS. Visualizza le playlist con intervallo di byte nella Registrazione automatica su Amazon S3 (streaming a bassa latenza).	✓		
CBR	Bitrate costante, un metodo di controllo della velocità per codificatori che mantiene un bitrate costante durante l'intera riproduzione di un video, indipendentemente da ciò che accade durante la trasmissione. È possibile aggiungere interruzioni all'azione per ottenere il bitrate desiderato, mentre i picchi possono essere quantizzati regolando la qualità della codifica in modo che corrisponda al bitrate desiderato. Consigliamo vivamente di utilizzare CBR anziché VBR.	✓	✓	
CDN	Rete per la distribuzione di contenuti, una soluzione distribuita geograficamente che ottimizza la distribuzione di contenuti come lo streaming video avvicinandoli al luogo in cui si trovano gli utenti.	✓		

Termine	Descrizione	LL	RT	Chat
Canale	Una risorsa IVS che memorizza la configurazione per lo streaming, tra cui un server di acquisizione, una chiave di streaming, un URL di riproduzione e opzioni di registrazione. Gli streamer utilizzano la chiave di flusso associata a un canale per avviare una trasmissione. Tutti i parametri e gli eventi generati durante una trasmissione sono associati a una risorsa del canale.	✓		
Tipo di canale	Determina la risoluzione e la frequenza fotogrammi per il canale. Consulta Tipi di canale nella Documentazione di riferimento delle API di streaming a bassa latenza IVS.	✓		
Log di chat	Un'opzione avanzata che può essere abilitata associando una configurazione di registrazione a una chat room.			✓
Chat room	Una risorsa IVS che memorizza la configurazione per una sessione di chat, incluse funzionalità opzionali come Revisione dei messaggi di chat e Log di chat. Consulta Passaggio 2: creazione di una chat room nella Guida introduttiva a IVS Chat.			✓
Composizione lato client	Utilizza un dispositivo host per mixare i flussi audio e video dei partecipanti alla fase e quindi li invia come flusso composito a un canale IVS. Ciò consente un maggiore controllo sull'aspetto della composizione a scapito di un maggiore utilizzo delle risorse del client e di un rischio maggiore che un problema relativo a fase host influisca sugli spettatori. Consulta anche Composizione lato server.	✓	✓	
CloudFront	Un servizio CDN fornito da Amazon.	✓		

Termine	Descrizione	LL	RT	Chat
CloudTrail	Un servizio AWS per la raccolta, il monitoraggio, l'analisi e la conservazione di eventi e attività degli account da AWS e fonti esterne. Vedi Registrazione delle chiamate API IVS con AWS. CloudTrail	✓	✓	✓
CloudWatch	Un servizio AWS per il monitoraggio delle applicazioni, la risposta ai cambiamenti delle prestazioni, l'ottimizzazione dell'uso delle risorse e la fornitura di informazioni sullo stato operativo. Puoi utilizzare CloudWatch per monitorare i parametri IVS; vedi Monitoraggio dello streaming in tempo reale di IVS e Monitoraggio dello streaming a bassa latenza di IVS.	✓	✓	✓
Composizione	Il processo di combinazione di flussi audio e video da più fonti in un unico flusso.	✓	✓	
Pipeline di composizione	Una sequenza di fasi di elaborazione necessarie per combinare più flussi e codificare il flusso risultante.	✓	✓	
Compressione	Codifica delle informazioni utilizzando un numero inferiore di bit rispetto alla rappresentazione originale. Qualsiasi compressione particolare è priva di perdita o con perdita. La compressione priva di perdita di dati riduce i bit, identificando ed eliminando la ridondanza statistica. Nessuna informazione viene persa nella compressione priva di perdita di dati. La compressione con perdita di dati riduce i bit, rimuovendo informazioni non necessarie o meno importanti.	✓	✓	
Piano di controllo (control-plane)	Memorizza informazioni sulle risorse IVS come canali, fasi o chat room e fornisce interfacce per la creazione e la gestione di tali risorse. È regionale (basato sulle regioni AWS).	✓	✓	✓

Termine	Descrizione	LL	RT	Chat
CORS	La funzionalità di condivisione delle risorse multiorigine definisce un metodo con cui le applicazioni Web dei clienti caricate in un dominio possono interagire con le risorse situate in un dominio differente, come bucket S3. L'accesso può essere configurato in base a intestazioni, metodi HTTP e domini di origine. Consulta Utilizzo delle funzionalità di condivisione di risorse multiorigine (CORS) - Amazon Simple Storage Service nella Guida per l'utente di Amazon Simple Storage Service.	✓		
Origine di immagini personalizzate	Un'interfaccia fornita dall'SDK di trasmissione IVSche consente a un'applicazione di fornire il proprio input di immagini anziché limitarsi alle fotocamere preimpostate.	✓	✓	
Piano dati	L'infrastruttura che trasporta i dati dall'acquisizione all'uscita. Funziona in base alla configurazione gestita nel piano di controllo e si limita a una regione AWS.	✓	✓	✓
Encoder, encoding	Il processo di conversione di contenuti video e audio in un formato digitale, adatto allo streaming. La codifica può essere basata su hardware o software.	✓	✓	
Evento	Una notifica automatica pubblicata da IVS al servizio di monitoraggio. AmazonEventBridge Un evento rappresenta una modifica dello stato o dello stato di salute di una risorsa di streaming, ad esempio uno stage o una pipeline di composizione. Vedi Utilizzo di Amazon EventBridge con streaming IVS a bassa latenza e Utilizzo di Amazon EventBridge con IVS Real-Time Streaming.	✓	✓	✓

Termine	Descrizione	LL	RT	Chat
FFmpeg	Un progetto software gratuito e open source costituito da una suite di librerie e programmi per la gestione di file e streaming video e audio. FFmpeg offre una soluzione multipiattaforma per registrare, convertire e trasmettere audio e video.	✓		
Streaming frammentato	Creato quando una trasmissione si disconnette e riconnette entro l'intervallo specificato nella configurazione di registrazione del canale. I flussi multipli risultanti vengono considerati un'unica trasmissione e uniti in un singolo flusso registrato. Consulta Unisci flussi frammentati nella Registrazione automatica su Amazon S3 (streaming a bassa latenza).	✓		
Frequenza fotogrammi	Un parametro di streaming per il numero di frame video trasmessi o ricevuti al secondo.	✓	✓	
HLS	HTTP Live Streaming (HLS), un protocollo di comunicazione di streaming bitrate adattivo basato su HTTP utilizzato per fornire streaming IVS agli spettatori.	✓		
Playlist HLS	Un elenco di segmenti multimediali che compongono uno streaming. Le playlist HLS standard sono composte da file multimediali di 10 secondi. HLS supporta anche playlist con intervallo di byte più granulari.	✓		
Host	Un partecipante all'evento in tempo reale che invia video e/o audio alla fase.		✓	
IAM	Identity and Access Management, un servizio AWS che consente agli utenti di gestire in modo sicuro le identità e l'accesso ai servizi e alle risorse AWS, incluso IVS.	✓	✓	✓

Termine	Descrizione	LL	RT	Chat
Acquisizione	Processo IVS per la ricezione di flussi video da un host o emittente per l'elaborazione o la distribuzione a spettatori o altri partecipanti.	✓	✓	
Server di acquisizione	Riceve i flussi video e li invia a un sistema di transcodifica, dove gli streaming vengono transmixati o transcodificati in HLS per essere consegnati agli spettatori. I server per l'importazione sono componenti IVS specifici che ricevono flussi per i canali, insieme a un protocollo di ingestione (RTMP, RTMPS). Consulta le informazioni sulla creazione di un canale in Guida introduttiva allo streaming a bassa latenza di IVS.		✓	
Video interlacciato	Trasmette e visualizza solo righe pari o dispari dei fotogrammi successivi per creare un raddoppio percepito della frequenza fotogrammi senza consumare ulteriore larghezza di banda. Si sconsiglia di utilizzare video interlacciati a causa di problemi di qualità video.	✓	✓	
JSON	JavaScript Object Notation, un formato di file a standard aperto che utilizza testo leggibile dall'uomo per trasmettere oggetti di dati costituiti da coppie attributo-valore e tipi di dati di array o altri valori serializzabili.	✓	✓	✓

Termine	Descrizione	LL	RT	Chat
Fotogramma chiave, fotogramma delta, intervallo di fotogrammi chiave	Il fotogramma chiave (noto anche come intracodificato o i-frame) è un fotogramma completo dell'immagine di un video. I fotogrammi successivi, i fotogrammi delta (detti anche fotogrammi previsti o p-frame), contengono solo le informazioni che sono state modificate. I fotogrammi chiave appariranno più volte all'interno di un flusso, a seconda dell'intervallo di fotogrammi chiave definito nell'encoder.	✓	✓	
Lambda	Un servizio AWS per l'esecuzione di codice (denominato funzioni Lambda) senza effettuare il provisioning di alcuna infrastruttura server. Le funzioni Lambda possono essere eseguite in risposta a eventi e richieste di chiamata o in base a una pianificazione. Ad esempio, IVS Chat utilizza le funzioni Lambda per consentire la revisione dei messaggi per una chat room.	✓	✓	✓
Latenza glass-to-glass, latenza	Un ritardo nel trasferimento dei dati. IVS definisce gli intervalli di latenza come: • Bassa latenza: meno di 3 sec • Latenza in tempo reale: inferiore a 300 ms La lass-to-glass latenza G si riferisce al ritardo tra il momento in cui una videocamera riprende un live streaming e il momento in cui lo stream appare sullo schermo dello spettatore.	✓	✓	
Codifica a livelli con simulcast	Consente la codifica e la pubblicazione simultanee di più flussi video con diversi livelli di qualità. Consulta Streaming adattivo: codifica a più livelli con Simulcast nelle ottimizzazioni dello streaming in tempo reale.		✓	

Termine	Descrizione	LL	RT	Chat
Gestore di revisione dei messaggi	Consente ai clienti di IVS Chat di esaminare/filtrare automaticamente i messaggi di chat degli utenti prima che vengano recapitati alla chat room. Viene abilitato associando una funzione Lambda a una chat room. Consulta Creazione di una funzione Lambda di Gestore di revisione dei messaggi di chat.			✓
Mixer	Una funzionalità degli SDK di trasmissione IVS per dispositivi mobili che prende più sorgenti audio e video e genera un'unica uscita. Supporta la gestione degli elementi video e audio sullo schermo che rappresentano sorgenti come fotocamere, microfoni, catture dello schermo e audio e video generati dall'applicazione. L'output può quindi essere trasmesso in streaming a IVS. Consulta Configurazione di una sessione di trasmissione per la combinazione in SDK di trasmissione IVS: guida alla combinazione (streaming a bassa latenza).	✓		
Streaming su più host	Combina i flussi provenienti da più host in un unico flusso. Può essere ottenuto utilizzando una composizione lato client o lato server. Lo streaming su più host consente scenari, come invitare gli spettatori su un palco per domande e risposte, competizioni tra host, chat video e host che conversano tra loro di fronte a un vasto pubblico.		✓	
Playlist multivariante	Un indice di tutte le varianti di streaming disponibili per una trasmissione.	✓		

Termine	Descrizione	LL	RT	Chat
OAC	Origin Access Control, un meccanismo per limitare l'accesso a un bucket S3, in modo che contenuti come uno stream registrato possano essere serviti solo tramite CDN. CloudFront	✓		
OBS	Open Broadcaster Software, software gratuito e open source per la registrazione video e lo streaming live. OBS offre un'alternativa (all'SDK di trasmissione IVS) per la pubblicazione per desktop. Gli streamer più sofisticati che hanno familiarità con OBS potrebbero preferirlo per le sue funzionalità di produzione avanzate, come le transizioni di scena, il mixaggio audio e la grafica di sovrapposizione.	✓	✓	
Partecipante	Un utente in tempo reale connesso alla fase come presentatore o spettatore.		✓	
Token dei partecipanti	Autentica un partecipante all'evento in tempo reale quando partecipa a un palco. Un token partecipante controlla anche se un partecipante può inviare video allo stage.		✓	
Token di riproduzione, coppia di chiavi di riproduzione	Un meccanismo di autorizzazione che consente ai clienti di limitare la riproduzione di video su canali privati. I token di riproduzione vengono generati da una coppia di chiavi di riproduzione. Una coppia di chiavi di riproduzione è la coppia di chiavi pubblica-privata utilizzata per firmare e convalidare il token di autorizzazione dello spettatore per la riproduzione. Consulta Creazione o importazione di una chiave di riproduzione in Configurazione dei canali privati e vedi gli endpoint della coppia di chiavi di riproduzione nel riferimento alle API a bassa latenza IVS.	✓		

Termine	Descrizione	LL	RT	Chat
URL di riproduzione	Identifica l'indirizzo utilizzato dallo spettatore per avviare la riproduzione di un canale specifico. Questo indirizzo può essere utilizzato a livello globale. IVS seleziona automaticamente la migliore posizione sulla rete globale di distribuzione dei contenuti IVS per ogni spettatore per la distribuz ione del video. Consulta le informazioni sulla creazione di un canale in Guida introduttiva allo streaming a bassa latenza di IVS.	✓		
Canale privato	Consente ai clienti di limitare l'accesso ai propri streaming utilizzando un meccanismo di autorizza zione basato su token di riproduzione. Consulta Flussi di lavoro per canali privati in Configurazione dei canali privati.	✓		
Video progressivo	Trasmette e visualizza tutte le linee di ogni fotogramma in sequenza. Si consiglia di utilizzar e il video progressivo durante tutte le fasi di una trasmissione.	✓	✓	
Quote	Il numero massimo possibile di risorse o operazion i del servizio IVS per il tuo account AWS. In altre parole, questi limiti sono per account AWS se non diversamente indicato. Tutte le quote vengono applicate per regione. Consulta endpoint e quote di Amazon Interactive Video Service in Guida di riferimento generale di AWS.	✓	✓	✓

Termine	Descrizione	LL	RT	Chat
Regioni	Consentono di accedere ai servizi AWS che risiedono fisicamente in un'area geografica specifica. Le regioni forniscono la tolleranza ai guasti, la stabilità e la resilienza e possono anche ridurre la latenza. Con le Regioni, è possibile creare risorse ridondanti che restano disponibili e non influenzate da un'interruzione a livello regionale. La maggior parte delle richieste di servizi AWS è associata a una particolare regione geografica. Le risorse create in una regione non esistono in qualsiasi altra regione, a meno che non si utilizzi in modo esplicito una funzionalità di replica offerta da un servizio AWS. Ad esempio, Amazon S3 supporta la replica tra regioni. Alcuni servizi, ad esempio IAM, non dispongono di risorse regionali.	✓	✓	✓
Risoluzione	Descrive il numero di pixel in un singolo fotogramma video, ad esempio, Full HD o 1080p definisce un fotogramma con 1920x1080 pixel.	✓	✓	
Utente root	Il proprietario dell'account AWS. L'utente root ha accesso completo a tutte le risorse e i servizi AWS in tale account.	✓	✓	✓
RTMP, RTMPS	Real-Time Messaging Protocol, uno standard di settore per la trasmissione di audio, video e dati su una rete. RTMPS è la versione sicura di RTMP, in esecuzione su una connessione Transport Layer Security (TLS/SSL).	✓	✓	

Termine	Descrizione	LL	RT	Chat
Bucket S3	Una raccolta di oggetti archiviati in Amazon S3. Molte policy, tra cui l'accesso e la replica, sono definite a livello di bucket e si applicano a tutti gli oggetti del bucket. Ad esempio, una trasmissione IVS viene archiviata come più oggetti in un bucket S3.	✓		
SDK	Software Development Kit, una raccolta di librerie per gli sviluppatori che creano applicazioni con IVS.	✓	✓	✓
Segmentazione dei selfie	Consente di sostituire lo sfondo in uno streaming live, utilizzando una soluzione specifica del client che accetta l'immagine della telecamera come input e restituisce una maschera con punteggi di affidabilità per ogni pixel dell'immagine, indicando se è in primo piano o sullo sfondo. Consulta Sostituzione dello sfondo in SDK di trasmissione IVS: filtri di fotocamere di terze parti (streaming in tempo reale).		✓	
Versione semantica	Un formato di versione sotto forma di Major.Minor.Patch. Le correzioni di bug che non influiscono sull'API incrementano la versione della patch, le aggiunte/modifiche alle API compatibili con le versioni precedenti incrementano la versione secondaria e le modifiche dell'API incompatibili con le versioni precedenti incrementano la versione principale.	✓	✓	✓

Termine	Descrizione	LL	RT	Chat
Composizione lato server	Utilizza un server IVS per mixare audio e video dei partecipanti alla fase e quindi invia questo video misto a un canale IVS per raggiungere un pubblico più ampio o per archiviarlo in un bucket S3. La composizione lato server riduce il carico del client, migliora la resilienza della trasmissione e consente un uso più efficiente della larghezza di banda. Consulta anche Composizione lato client.		✓	
Quote del servizio	Un servizio che consente di gestire le quote per numerosi servizi da un'unica posizione. Oltre a cercare i valori delle quote, nella console Service Quotas è possibile richiedere anche un aumento delle quote.	✓	✓	✓
Ruolo collegato al servizio	Un unico tipo di ruolo IAM collegato direttamente a un servizio AWS. I ruoli collegati ai servizi sono creati automaticamente da IVS e includono tutte le autorizzazioni richieste dal servizio per eseguire chiamate agli altri servizi AWS per tuo conto, per esempio, per accedere a un bucket S3. Consulta Utilizzo dei ruoli collegati ai servizi per IVS in Sicurezza IVS.	✓		
Stage	Una risorsa IVS che rappresenta uno spazio virtuale in cui i partecipanti agli eventi in tempo reale possono scambiarsi video in tempo reale. Consulta Creazione di una fase in Giuda introduttiva allo streaming in tempo reale di IVS.		✓	
Sessione di fase	Inizia quando il primo partecipante si unisce a una fase e termina pochi minuti dopo che l'ultimo ha smesso di pubblicare nella fase. Una fase di lunga durata può avere più sessioni nel corso della sua durata.		✓	

Termine	Descrizione	LL	RT	Chat
Flusso	Dati che rappresentano contenuti video o audio inviati continuamente da un'origine a una destinazione.	✓	✓	
Chiave di streaming	Un identificatore assegnato da IVS quando si crea un canale, utilizzato per autorizzare lo streaming al canale. Tratta la chiave di streaming come un segreto, poiché consente a chiunque di trasmettere in streaming al canale. Consulta Guida introduttiva allo streaming a bassa latenza di IVS.	✓		
Starvation di flussi	Ritardo o interruzione della trasmissione dello streaming a IVS. Si verifica quando IVS non riceve la quantità di bit prevista che il dispositivo di codifica avrebbe dovuto inviare in un determinato intervallo di tempo. In caso di interruzione dello streaming, si verifica un evento di starvation di flussi. Dal punto di vista dei visualizzatori, starvation di flussi può apparire come ritardo, buffering o blocco di un video. Starvation di flussi può essere breve (meno di 5 secondi) o lunga (diversi minuti), a seconda della situazione specifica che ha provocato la starvation. Consulta Cos'è la starvation di flussi nelle Domande frequenti sulla risoluzione dei problemi.	✓	✓	
Streamer	Una persona o un dispositivo che invia un streaming video o audio a IVS.	✓	✓	
Sottoscrittore	Un partecipante all'evento in tempo reale che riceve video e/o audio degli organizzatori. Consulta Cos'è lo streaming in tempo reale IVS.		✓	

Termine	Descrizione	LL	RT	Chat
Tag	Un tag è un'etichetta che viene assegnata a una risorsa AWS. I tag consentono di identificare e organizzare le risorse AWS. Nella pagina iniziale della documentazione IVS, consulta "Applicazione di tag" in qualsiasi documentazione dell'API IVS (per lo streaming in tempo reale, lo streaming a bassa latenza o la chat).	✓	✓	✓
Filtri di fotocamere di terze parti	Componenti software che possono essere integrati con SDK di trasmissione IVS per consentire a un'applicazione di elaborare le immagini prima di fornirle un SDK di trasmissione come fonte di immagini personalizzata. Un filtro di terze parti può elaborare le immagini dalla fotocamera, applicare un effetto filtro, ecc.	✓	✓	
Anteprima	Un'immagine di dimensioni ridotte presa da uno streaming. Per impostazione predefinita, le miniature vengono generate ogni 60 secondi, ma è possibile configurare un intervallo più breve. La risoluzione delle miniature dipende dal tipo di canale. Consulta Registrazione di contenuti in Registrazione automatica su Amazon S3 (streamin g a bassa latenza).	✓		

Termine	Descrizione	LL	RT	Chat
Metadati temporizzati	Metadati legati a timestamp specifici all'interno di uno streaming. Possono essere aggiunti a livello di codice utilizzando l'API IVS e vengono associati a fotogrammi specifici. Ciò garantisce che tutti gli spettatori ricevano i metadati nello stesso punto dello streaming. I metadati temporizzati possono essere utilizzati per attivare azioni sul client, come l'aggiornamento delle statistiche della squadra durante un evento sportivo. Consulta Incorporamento di metadati all'interno di un flusso video.	✓		
Transcodifica	Converte video e audio da un formato all'altro. Un flusso in ingresso può essere transcodificato in un formato diverso a più bitrate e risoluzioni in modo da supportare una serie di dispositivi di riproduzione e diverse condizioni di rete.	✓	✓	
Transmuxing	Un semplice riconfezionamento di un flusso acquisito su IVS, senza ricodifica del flusso video. "Transmux" è l'abbreviazione di transcode multiplexing, un processo che cambia il formato di un file audio e/o video mantenendo alcuni o tutti i flussi originali. Il trasmuxing esegue la conversione in un formato container diverso senza modificare il contenuto del file. Diverso dalla transcodifica.	✓	✓	

Termine	Descrizione	LL	RT	Chat
Flussi di variante	Un insieme di codifiche della stessa trasmissione in diversi livelli di qualità distinti. Ogni flusso di variante è codificato come riproduzione HLS separata. Un indice dei flussi di variante disponibili viene definito riproduzione multivariante. Dopo che il lettore IVS ha ricevuto una riproduzione multivariante da IVS, può scegliere tra i flussi di variante durante la riproduzione, passando da uno all'altro senza interruzioni al variare delle condizioni della rete.	✓		
VBR	Variable Bitrate, un metodo di controllo della velocità per codificatori che utilizza un bitrate dinamico che cambia durante la riproduzione, a seconda del livello di dettaglio richiesto. Non usare VBR per motivi di qualità video; usa invece CBR.	✓	✓	

Termine	Descrizione	LL	RT	Chat
Vista	Una sessione di visualizzazione unica che sta attivamente scaricando o riproducendo un video. Le visualizzazioni sono la base per la quota di visualizzazioni simultanee. Una vista inizia quando una sessione di visualizz azione inizia la riproduzione video. Una vista termina quando una sessione di visualizzazione interrompe la riproduzione video. La riproduzione è l'unico indicatore dello spettatore; l'euristica del coinvolgimento come i livelli audio, la messa a fuoco delle schede del browser e la qualità video non vengono prese in considerazione. Quando si contano le viste, IVS non considera la legittimi tà dei singoli spettatori né tenta di deduplicare le visualizzazioni localizzate, ad esempio più lettori video su un singolo computer. Consulta Altre quote in Service Quotas (streaming a bassa latenza).	✓		
Visualizzatore	Una persona che riceve uno streaming da IVS.	✓		
WebRTC	Web Real-Time Communication, un progetto open source che fornisce ai browser Web e alle applicazi oni mobili comunicazioni in tempo reale. Consente alla comunicazione audio e video di funzionar e all'interno delle pagine Web permettendo la peer-to-peer comunicazione diretta, eliminando la necessità di installare plug-in o scaricare app native. Le tecnologie alla base di WebRTC sono implementate come standard web aperto e sono disponibili come JavaScript normali API in tutti i principali browser o come librerie per client nativi, come Android e iOS.	✓	✓	

Termine	Descrizione	LL	RT	Chat
FRUSTA	WebRTC-HTTP Ingestion Protocol, un protocollo basato su HTTP che consente l'inserimento di contenuti basato su WebRTC in servizi di streaming e/o CDN. WHIP è una bozza IETF sviluppata per standardizzare l'ingestione di WebRTC. WHIP consente la compatibilità con software come OBS, offrendo un'alternativa (all'IVS broadcast SDK) per il desktop publishing. Gli streamer più sofisticati che hanno familiarità con OBS potrebbero preferirlo per le sue funzionalità di produzione avanzate, come le transizioni di scena, il mixaggio audio e la grafica di sovrapposizione WHIP è utile anche in situazioni in cui l'utilizzo dell'SDK di trasmissione IVS non è fattibile o preferibile. Ad esempio, nelle configurazioni che coinvolgono codificatori hardware, l'SDK di trasmissione IVS potrebbe non essere un'opzione. Tuttavia, se l'encoder supporta WHIP, è comunque possibile pubblicare direttamente dall'encoder su IVS. Vedi OBS e WHIP Support.		✓	
WSS	WebSocket Secure, un protocollo per stabilire WebSockets una connessione TLS crittografata. Viene utilizzato per la connessione agli endpoint di IVS Chat. Consulta Passaggio 4: inviare e ricevere il primo messaggio in Guida introduttiva alla chat di IVS.			✓

Cronologia dei documenti (streaming in tempo reale)

Modifiche alla Guida per l'utente dello streaming in tempo reale

Modifica	Descrizione	Data
Supporto OBS e WHIP	Aggiunta una nuova pagina. Questo documento spiega come utilizzare codificat ori compatibili con WHIP come OBS per pubblicare su IVS lo streaming in tempo reale. WHIP (WebRTC-H TTP Ingestion Protocol) è una bozza IETF sviluppata per standardizzare l'ingestione di WebRTC.	6 febbraio 2024
SDK di trasmissione: Android 1.14.1, iOS 1.14.1, Web 1.8.0	Numero di versione aggiornato e collegamenti agli artefatti per la nuova versione, nelle guide SDK di real-time-streaming trasmissione: Android,iOS e Web. Nella pagina di destinazi one della documentazione di Amazon IVS sono stati aggiornati i collegamenti ai riferimenti SDK di trasmissione in modo da puntare alla nuova versione. Consulta anche le Note di rilascio di Amazon IVS per questa versione. Per la Guida Android, abbiamo aggiunto un nuovo problema	1 febbraio 2024

	noto (dimensioni video inferiori a 176x176). Per la Guida Web, abbiamo aggiunto un nuovo problema noto. La soluzione alternativa consiste nel limitare la risoluzione video a 720p quando si richiama o. `getUserMedia getDisplayMedia` In Real-Time Streaming Optimizations abbiamo aggiornato la configurazione della codifica a strati con Simulcast; ora questa opzione è disabilitata per impostazione predefinita.	
SDK di trasmissione: Android 1.13.4, iOS 1.13.4, Web 1.7.0	Numero di versione aggiornato e collegamenti agli artefatti per la nuova versione, nelle guide SDK di real-time-streaming trasmissione: Android,iOS e Web. Nella pagina di destinazione della documentazione di Amazon IVS sono stati aggiornati i collegamenti ai riferimenti SDK di trasmissione in modo da puntare alla nuova versione. Consulta anche le Note di rilascio di Amazon IVS per questa versione.	3 gennaio 2024

Glossario IVS	Ha esteso il glossario, coprendo i termini IVS in tempo reale, a bassa latenza e di chat.	20 dicembre 2023
Stage Health: nuove CloudWatch metriche	La metrica PacketLoss (Stage) è stata ribattezzata DownloadPacketLoss (Stage) e sono state rilasciate CloudWatch metriche aggiuntive per lo streaming in tempo reale IVS: • DownloadPacketLoss (Stage, Partecipante) • DroppedFrames (Stage, Partecipante) • SubscribeBitrate (Fase, Partecipante,) MediaType Consulta Monitoraggio dello streaming in tempo reale di Amazon IVS.	7 dicembre 2023

Policy gestite da IAM	Sono state aggiunte due politiche gestite, IVS ReadOnlyAccess e IVS. FullAccess Vedere: • La nuova sezione sulle Policy gestite per Amazon IVS nella pagina Sicurezza. • Modifiche al Passaggio 3: configurazione delle autorizzazioni IAM in Guida introduttiva allo streaming a bassa latenza di IVS.	5 dicembre 2023
SDK di trasmissione: Android 1.13.2 e iOS 1.13.2	Numero di versione e link agli artefatti aggiornati per la nuova versione, nelle guide SDK di real-time-streaming trasmissione: Android e iOS. Nella pagina di destinazione della documentazione di Amazon IVS sono stati aggiornati i collegamenti ai riferimenti SDK di trasmissione in modo da puntare alla nuova versione. Consulta anche le Note di rilascio di Amazon IVS per questa versione.	4 dicembre 2023

SDK di trasmissione: Android 1.13.1	Numero di versione e link agli artefatti aggiornati per la nuova versione, nella guida SDK di real-time-streaming trasmissione: Android. Nella pagina di destinazione della documentazione di Amazon IVS sono stati aggiornati i collegamenti ai riferimenti SDK di trasmissione in modo da puntare alla nuova versione. Consulta anche le Note di rilascio di Amazon IVS per questa versione.	21 novembre 2023
Service Quotas	Modificata la "risoluzione di pubblicazione dei partecipanti" da 1080p a 720p.	18 novembre 2023

SDK di trasmissione: Android 1.13.0 e iOS 1.13.0	Numero di versione e link agli artefatti aggiornati per la nuova versione, nelle guide SDK di real-time-streaming trasmissione: Android e iOS. Nella pagina di destinazione della documentazione di Amazon IVS sono stati aggiornati i collegamenti ai riferimenti SDK di trasmissione in modo da puntare alla nuova versione. Consulta anche le Note di rilascio di Amazon IVS per questa versione. Abbiamo anche apportato vari aggiornamenti alla sezione Ottimizzazioni dello streaming. Tra le altre cose, la funzionalità "Streaming adattativo: codifica a livelli con simulcast " ora richiede un consenso esplicito ed è supportata solo nelle versioni recenti dell'SDK.	17 novembre 2023

Registrazione composita	Sono state apportate le modifiche seguenti: • È stata aggiunta una pagina Registrazione composita per questa nuova funzionalità. • La Guida introduttiva allo streaming in tempo reale di IVS è stata aggiornata con gli endpoint S3 nella policy in "Configurazione delle autorizzazioni IAM". • Aggiornata la sezione Service Quotas con le quote tariffarie di chiamata per i nuovi endpoint.	16 novembre 2023

Composizione lato server (SSC)	La composizione lato server di IVS consente ai client di affidare la composizione e la trasmissione di una fase IVS a un servizio gestito da IVS. La composizione lato server e la trasmissione RTMP a un canale vengono richiamate tramite gli endpoint del piano di controllo (control-plane) IVS nella regione di origine della fase. Vedere: • Guida introduttiva: abbiamo aggiunto gli endpoint SSC alla policy in "Configur azione delle autorizzazioni IAM". • Utilizzo di Amazon EventBridge con IVS: abbiamo aggiunto nuove metriche. • Composizione lato server: questo nuovo documento include una panoramica e istruzioni di configurazione. • Service Quotas: abbiamo aggiunto nuovi limiti di frequenza delle chiamate e altre quote. Consulta anche: • Modifiche riportate di seguito in Modifiche alla documentazione di riferimen	16 novembre 2023

	to delle API di streaming in tempo reale IVS. • Modifiche riportate in Cronologia dei documenti (streaming a bassa latenza).	
SDK di trasmissione IVS	In Panoramica dell'SDK di trasmissione, abbiamo aggiornato Requisiti della piattaforma > Piattaforme native per chiarire quali versioni dell'SDK sono supportate e abbiamo aggiunto "Browser per dispositi vi mobili (iOS e Android)". Nella Guida per la trasmissi one Web, abbiamo aggiunto "Limitazioni Web per i dispositi vi mobili".	9 novembre 2023
SDK di trasmissione IVS	Abbiamo aggiunto una nuova pagina in Filtri per fotocamere di terze parti.	9 novembre 2023
Guida introduttiva allo streaming in tempo reale di IVS	Abbiamo aggiornato le procedure in Impostazione delle autorizzazioni IAM.	20 ottobre 2023
Monitoraggio dello streaming in tempo reale	In CloudWatch Metrics: IVS Real-Time Streaming, abbiamo aggiunto valori di esempio per le dimensioni.	17 ottobre 2023

SDK di trasmissione: guida per il Web	Abbiamo apportato diverse modifiche a Monitoraggio dello stato di silenziamento dei contenuti multimediali dei partecipanti remoti.	17 ottobre 2023
SDK di trasmissione: Web 1.6.0	Numero di versione e link agli artefatti aggiornati per la nuova versione, nella guida SDK di real-time-streaming trasmissione: Web. La pagina di destinazione della documentazione di Amazon IVS rimanda alla versione attuale della documentazione di riferimento dell'SDK di trasmissione. Consulta anche le Note di rilascio di Amazon IVS per questa versione. Nella Guida Web, in «Recupera un file MediaStream da un dispositivo», abbiamo eliminato anche le due righe; la migliore pratica è quella di specificarle solo. `max` `ideal` In Ottimizzazione dello streaming in tempo reale, abbiamo aggiunto una nuova sezione, Ottimizzazione del bitrate audio e supporto stereo.	16 ottobre 2023

Stage Health: nuove CloudWatch metriche	CloudWatch Metriche rilasciate per lo streaming in tempo reale di IVS. Consulta Monitoraggio dello streaming in tempo reale di Amazon IVS.	12 ottobre 2023
SDK di trasmissione: Android 1.12.1	Numero di versione e link agli artefatti aggiornati per la nuova versione, nella guida SDK di real-time-streaming trasmissione: Android. È stata inoltre aggiunta una nuova sezione, Utilizzo dei microfoni Bluetooth. La pagina di destinazione della documentazione di Amazon IVS rimanda alla versione attuale della documentazione di riferimento dell'SDK di trasmissione. Consulta anche le Note di rilascio di Amazon IVS per questa versione.	12 ottobre 2023

SDK di trasmissione: Web 1.5.2	Numero di versione e collegamenti agli artefatti aggiornati per la nuova versione, nella guida SDK di real-time-streaming trasmissione: Web. La pagina di destinazione della documentazione di Amazon IVS rimanda alla versione attuale della documentazione di riferimento dell'SDK di trasmissione. Consulta anche le Note di rilascio di Amazon IVS per questa versione.	14 settembre 2023
Guida introduttiva allo streaming in tempo reale di IVS	In Android > Installa l'SDK di trasmissione, è stata aggiunta l'associazione dei dati.	12 settembre 2023
Gestione degli errori dell'SDK di trasmissione	Aggiunte le sezioni "Gestione degli errori" alle guide all'SDK di trasmissione: Web, Android e iOS.	12 settembre 2023
Guida introduttiva allo streaming in tempo reale di IVS	In Distribuisci i token dei partecipanti, è stata aggiunta una nota Importante sulla mancata creazione di funzionalità basate sul formato del token corrente.	1 settembre 2023
Guida introduttiva allo streaming in tempo reale di IVS	In Configura le autorizzazioni IAM, è stato aggiornato il set di autorizzazioni.	31 agosto 2023

SDK di trasmissione: Web 1.5.1, Android 1.12.0 e iOS 1.12.0	Numero di versione aggiornato e collegamenti agli artefatti per la nuova versione, nelle guide SDK di real-time-streaming trasmissione: Web, Android e iOS. Nella pagina di destinazione della documentazione di Amazon IVS sono stati aggiornati i collegamenti ai riferimenti SDK di trasmissione in modo da puntare alla nuova versione. Consulta anche le Note di rilascio di Amazon IVS per questa versione.	23 agosto 2023

Lancio dello streaming in tempo reale	Questa release comprende le principali modifiche alla documentazione. Abbiamo rinominato la documentazione precedente in Streaming a bassa latenza IVS e pubblicato una nuova documentazione Streaming in tempo reale IVS. La pagina iniziale della documentazione IVS ora dispone di sezioni separate per lo streaming in tempo reale e lo streaming a bassa latenza. Ogni sezione ha una propria Guida per l'utente e la Documentazione di riferimento delle API. Per altre modifiche alla documentazione, consulta Cronologia dei documenti (streaming a bassa latenza).	7 agosto 2023

SDK di trasmissione: Web 1.5.0, Android 1.11.0 e iOS 1.11.0	Aggiornato il numero di versione e i collegamenti degli artefatti per la nuova versione nelle guide all'SDK di trasmissione: Web, Android e iOS. Nella pagina di destinazione della documentazione di Amazon IVS sono stati aggiornati i collegamenti ai riferimenti SDK di trasmissione in modo da puntare alla nuova versione. Consulta anche le Note di rilascio di Amazon IVS per questa versione.	7 agosto 2023

Modifiche alla Documentazione di riferimento delle API di streaming in tempo reale IVS

Modifiche alle API	Descrizione	Data
Registrazione composita	Abbiamo aggiunto 4 StorageConfiguration endpoint e 7 oggetti (DestinationDetail,, S3, S3Detail RecordingConfiguration, S3DestinationConfiguration,). StorageConfiguration StorageConfiguration StorageConfigurationSummary Abbiamo modificato 3 oggetti (Composizione, Destinazione,). DestinationConfiguration Ciò influisce sulla GetComposition risposta, sulla StartComposition richiesta e sulla risposta.	16 novembre 2023

Modifiche alle API	Descrizione	Data
Composizione lato server	Abbiamo aggiunto 8 Composizione e EncoderConfiguration punti finali e 11 oggetti (ChannelDestinationConfiguration, Composizione, CompositionSummary, Destinazione DestinationConfiguration DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration, LayoutConfiguration, e Video).	16 novembre 2023
Integrità della fase: nuovi dati sui partecipanti	Sono stati aggiunti sei campi all'oggetto Partecipante: `browserName` , `browserVersion` , `ispName`, `osName`, `osVersion` , e `sdkVersion` . Ciò influisce sulla GetParticipant risposta.	12 ottobre 2023
Token dei partecipanti	Aggiunta una nota Importante sulla mancata creazione di funzionalità basate sul formato del token corrente.	1 settembre 2023
Lancio dello streaming in tempo reale IVS	Questa release comprende le principali modifiche alla documentazione. Abbiamo rinominato la documentazione precedente in Streaming a bassa latenza IVS e pubblicato una nuova documentazione Streaming in tempo reale IVS. La pagina iniziale della documentazione IVS ora dispone di sezioni separate per lo streaming in tempo reale e lo streaming a bassa latenza. Ogni sezione ha una propria Guida per l'utente e la Documentazione di riferimento delle API. Documentazione di riferimento delle API di streaming in tempo reale IVS fa parte della documentazione di streaming in tempo reale di IVS. In precedenza si chiamava Documentazione di riferimento delle API della fase IVS. La sua storia precedente è descritta in Cronologia dei documenti (streaming a bassa latenza).	7 agosto 2023

Note di rilascio (streaming in tempo reale)

6 febbraio 2024

Supporto OBS e WHIP

IVS può essere utilizzato con encoder compatibili con WHIP come OBS per pubblicare su IVS lo streaming in tempo reale. WHIP (WebRTC-HTTP Ingestion Protocol) è una bozza IETF sviluppata per standardizzare l'ingestione di WebRTC. Vedi la nuova pagina su OBS e WHIP Support.

1 febbraio 2024

SDK Amazon IVS Broadcast: Android 1.14.1, iOS 1.14.1, Web 1.8.0 (streaming in tempo reale)

Piattaforma	Download e modifiche
SDK per la trasmissione Web 1.8.0	Documentazione di riferimento: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • La codifica a più livelli con simulcast è ora disabilitata per impostazione predefinita. • È stato risolto un problema per cui un'istanza Stage non si disconnetteva correttamente quando uno stage veniva eliminato o quando un partecipante veniva disconnesso dal server. L'SDK ora emette un `STAGE_CONNECTION_STATE_CHANGED` evento con uno stato di `DISCONNECTED` (anziché e then). `ERRORED` `CONNECTING` • Problema risolto per cui la pubblicazione non riusciva quando si aggiornava la strategia con tracce audio o video vuote.

Piattaforma	Download e modifiche
Android Broadcast SDK 1.14.1	Documentazione di riferimento: https://aws.github.io/ /1.14.1/android amazon-ivs-broadcast-docs • La codifica a più livelli con simulcast è ora disabilitata per impostazione predefinita. • Aggiornato `libWebRTC` da M108 a M119. • Risolti diversi arresti anomali per migliorare la stabilità generale. • È stato aggiunto il supporto per la pubblicazione stereo. Questo può essere abilitato tramite l'`StageAudioConfiguration` oggetto. • Risolto un bug che causava un feed nero da parte dei partecipanti dopo aver partecipato a una sessione. • `libWebRTC` Riferimenti interni aggiornati per evitare conflitti tra simboli quando altre `libWebRTC` versioni sono incluse nella stessa applicazione host.

Piattaforma	Download e modifiche
SDK di trasmissione iOS 1.14.1	Scaricalo per lo streaming in tempo reale: https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip Documentazione di riferimento: https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios • La codifica a più livelli con simulcast è ora disabilitata per impostazione predefinita. • Aggiornato `libWebRTC` da M108 a M119. • Risolti diversi arresti anomali per migliorare la stabilità generale. • È stato aggiunto il supporto per la pubblicazione stereo. Questo può essere abilitato tramite`IVSLocalStageStreamAudioConfiguration` . • Risolto un crash che si verificava quando si abilitava la modalità solo audio per gli altri partecipanti. • TTV migliorato e dimensione binaria ridotta.

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
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

Dimensione dell'SDK di trasmissione: iOS

Architettura	Dimensione compressa	Dimensione non compressa
arm64	3.350 MB	7.790 MB

3 gennaio 2024

SDK Amazon IVS Broadcast: Android 1.13.4, iOS 1.13.4, Web 1.7.0 (streaming in tempo reale)

Piattaforma	Download e modifiche
SDK per la trasmissione Web 1.7.0	Documentazione di riferimento: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • Migliorato time-to-video per gli abbonati che si iscrivono ai stage. • La `minAudioBitrateKbps` proprietà è stata rimossa (non era utilizzata). • Ripristino della rete migliorato durante interruzioni o modifiche a Internet.
Android Broadcast SDK 1.13.4	Documentazione di riferimento: https://aws.github.io/ /1.13.4/android amazon-ivs-broadcast-docs • StageAudioConfiguration ora supporta l'impostazione se la cancellazione dell'eco deve essere abilitata.
SDK di trasmissione iOS 1.13.4	Scaricalo per lo streaming in tempo reale: https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip

Piattaforma	Download e modifiche
	Documentazione di riferimento: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.4/ios • Su iOS, abbiamo migliorato il motore audio per la registrazione e la riproduzione con particolare attenzione alla stabilità e alla recuperabilità. Ciò migliora il supporto per le modifiche di percorso durante l'uso, migliora il recupero della batteria nei casi limite e riduce la quantità di blocchi del thread principale. • È stato risolto un problema per cui il microfono poteva rimanere attivo anche dopo essere stato scollegato da un palco, lasciando acceso l'indicatore di privacy iOS. (L'SDK non elaborava l'audio in entrata in quel momento.)

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
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	5.53 MB	14.214 MB

Dimensione dell'SDK di trasmissione: iOS

Architettura	Dimensione compressa	Dimensione non compressa
arm64	3,45 MB	7,84 MB

7 dicembre 2023

Nuove metriche CloudWatch

Abbiamo rinominato la metrica PacketLoss (Stage) in DownloadPacketLoss (Stage). Abbiamo anche rilasciato CloudWatch metriche aggiuntive per lo streaming in tempo reale di IVS:

- DownloadPacketLoss (Stage, Partecipante)
- DroppedFrames (Stage, Partecipante)
- SubscribeBitrate (Fase, Partecipante,) MediaType

Consulta Monitoraggio dello streaming in tempo reale di Amazon IVS per i dettagli.

4 dicembre 2023

SDK di trasmissione Amazon IVS: Android 1.13.2 e iOS 1.13.2 (streaming in tempo reale)

Piattaforma	Download e modifiche
Tutti i dispositivi mobili (Android e iOS)	• La configurazione di soppressione del rumore è disponibile per gli sviluppatori che possono abilitare/disabilitare per la pubblicazione.
SDK di trasmissione per Android 1.13.2	Documentazione di riferimento: https://aws.github.io/ /1.13.2/android amazon-ivs-broadcast-docs

Piattaforma	Download e modifiche
	• È stato migliorato il tempo necessario per caricare il video (TTV) quando si entra nella prima fase in una sessione.
SDK di trasmissione per iOS 1.13.2	Scarica per lo streaming in tempo reale: https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip Documentazione di riferimento: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/ios • Nessuna modifica nell'SDK in tempo reale.

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
arm64-v8a	5,177 MB	13,01 MB
armeabi-v7a	4,485 MB	9,045 MB
x86_64	5,352 MB	13,808 MB
x86	5,547 MB	14,192 MB

Dimensione dell'SDK di trasmissione: iOS

Architettura	Dimensione compressa	Dimensione non compressa
arm64	3,45 MB	7,82 MB

21 novembre 2023

SDK di trasmissione Amazon IVS: Android 1.13.1 (streaming in tempo reale)

Piattaforma	Download e modifiche
SDK di trasmissione per Android 1.13.1	Documentazione di riferimento: https://aws.github.io/ /1.13.1/android amazon-ivs-broadcast-docs • Risolto un problema che causava un arresto anomalo quando si usciva, si rilasciava e si rientrava rapidamente nello stesso livello.

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
arm64-v8a	5,177 MB	13,102 MB
armeabi-v7a	4,485 MB	9,046 MB
x86_64	5,353 MB	13,809 MB
x86	5,547 MB	14,192 MB

17 novembre 2023

SDK di trasmissione Amazon IVS: Android 1.13.0 e iOS 1.13.0 (streaming in tempo reale)

Piattaforma	Download e modifiche
Tutti i dispositivi mobili (Android e iOS)	• Ottimizzazioni dello streaming aggiornato. Tra le altre cose, la funzionalità "Streaming adattativo: codifica a livelli con simulcast" ora richiede un consenso esplicito ed è supportat a solo nelle versioni recenti dell'SDK. • È stata migliorata la stabilità degli stage riducendo il numero di arresti anomali. • È stato migliorato il tempo necessario per caricare il video (TTV) quando si entra in una fase. • È stata migliorata l'esperienza con i dispositi vi Bluetooth. • È stato ottimizzato l'utilizzo della CPU e della memoria degli SDK e sono state ridotte le dimensioni della libreria. • È stata aggiunta la classe `StageAudioManager` , che può essere utilizzata per impostare i parametri di acquisizione e riproduzione audio, incluse le preimpost azioni per la comunicazione vocale, la riproduzione multimediale e altro ancora. Per maggiori dettagli, consulta la nuova pagina, SDK di trasmissione IVS: modalità audio per dispositivi mobili. • È stata aggiunta una nuova funzione `requestQualityStats` per visualizzare eventi strutturati di qualità dalle statistiche WebRTC.

Piattaforma	Download e modifiche
	• È stata aggiunta una nuova funzione per aggiornare il bitrate audio. È impostata su oggetti `LocalStageStream` come la configurazione video, ma tramite un nuovo oggetto di configurazione audio.

Piattaforma	Download e modifiche

SDK di trasmissione per Android 1.13.0	Documentazione di riferimento: https://aws.github.io/ /1.13.0/android amazon-ivs-broadcast-docs
	
	• Tutti i metodi dell'interfaccia `StageRenderer` sono ora facoltativi.
	• È stato aggiunto il supporto all'anteprima basata su `Surfaceview` per prestazioni migliori. I metodi `getPreview` esistenti in `Session` e `StageStream` continuano a restituire una sottoclasse di`TextureView`, ma ciò potrebbe cambiare in una futura versione dell'SDK.
	  • Se l'applicazione dipende in modo specifico da `TextureView`, è possibile continuare senza apportare modifiche. Puoi anche passare da `getPreview` a `getPreviewTextureView` per prepararti all'eventuale modifica di ciò che restituisce il `getPreview` predefinito.
	  • Se la tua applicazione non richiede `TextureView` in modo specifico, ti consigliamo di passare a `getPreviewSurfaceView` per ridurre l'utilizzo della CPU e della memoria.
	• L'SDK ora implementa un nuovo tipo di anteprima denominato `ImagePreviewSurfaceTarget` che funziona con l'oggetto Android Surface fornito dall'applicazione. Non è una sottoclasse di Android View, che offre una maggiore flessibilità.
	• È stato risolto il caso in cui la richiamata di `onFrame` per il partecipante remoto veniva

Piattaforma	Download e modifiche
	effettuata nel momento sbagliato con la dimensione sbagliata. • `SurfaceSource # getInputSurface` ora è annotato con `@Nullable` . Il codice dovrebbe controllarlo prima di utilizzarlo. • Aggiunti `UserId` e `attributes a ParticipantInfo` . Le proprietà `UserId` e `attributes` sono incorporate nel token e le applicazioni possono recuperarle tramite `ParticipantInfo` ogni volta che un partecipante si unisce. • L'acquisizione da fotocamera e il rendering in anteprima ora sono impostati in automatic o su 720 x 1280 o con la risoluzione di pubblicazione (a seconda di quale sia maggiore) su 15 fps. È possibile regolare la risoluzione e/o gli fps utilizzando `StageVide oConfiguration # setCamera CaptureQuality` . • La `IllegalArgumentException` generata durante l'impostazione delle proprietà di configurazione ora include il valore fornito nel messaggio di eccezione.

Piattaforma	Download e modifiche
SDK di trasmissione per iOS 1.13.0	Scarica per lo streaming in tempo reale: https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentazione di riferimento: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.0/ios • È stato risolto il problema per cui l'SDK non modifica la configurazione video se la configurazione video viene aggiornata prima della pubblicazione. • È stata incorporata la correzione di Google per una vulnerabilità di sicurezza di LibVPX (CVE-2023-5217). (Tieni presente che l'SDK Android non ha richiesto alcuna modifica per questo problema.) • Le applicazioni che utilizzano altre librerie che includono `libWebRTC` non avranno più conflitti con l'SDK di trasmissione IVS. • Tutti i metodi del protocollo `IVSStageRenderer` sono ora contrassegnati con `@optional` . • I microfoni e le fotocamere restituiti dai nostri SDK ora hanno un ordine di ordinamento garantito, come documentato negli SDK stessi. • Ora più fotocamere possono avere un valore `true` per la rispettiva proprietà `isDefault` , una per ogni posizione determinata dal sistema operativo. • Aggiunto `IVSStageAudioManager` , che consente un controllo preciso sul `AVAudioSession` sottostante per

Piattaforma	Download e modifiche
	consentire una più ampia varietà di casi d'uso della funzionalità Stages. • È stato aggiunto `UserId` a `ParticipantInfo` .

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
arm64-v8a	5,17 MB	13 MB
armeabi-v7a	4,48 MB	9,04 MB
x86_64	5,35 MB	13,80 MB
x86	5,54 MB	14,18 MB

Dimensione dell'SDK di trasmissione: iOS

Architettura	Dimensione compressa	Dimensione non compressa
arm64	3,45 MB	7,84 MB

16 novembre 2023

Registrazione composita

Questa nuova funzionalità consente la registrazione della vista composita di una fase IVS su un bucket Amazon S3. Per ulteriori informazioni, consultare:

- Registrazione composita: questa è una nuova pagina.
- Guida introduttiva allo streaming in tempo reale di IVS: abbiamo aggiunto gli endpoint S3 alla policy riportata in "Configurazione delle autorizzazioni IAM".

- Service Quotas: sono state aggiunte le quote tariffarie di chiamata per i nuovi endpoint.
- Riferimento all'API IVS Real-Time Streaming: abbiamo aggiunto 4 StorageConfiguration endpoint e 7 oggetti (DestinationDetail,, S3, S3Detail, S3 RecordingConfigurationDestinationConfiguration,,). StorageConfiguration StorageConfiguration StorageConfigurationSummary Abbiamo anche modificato 3 oggetti (Composizione, Destinazione, DestinationConfiguration); ciò influisce sulla risposta, sulla richiesta e sulla GetComposition risposta. StartComposition

16 novembre 2023

Composizione lato server

La composizione lato server di IVS consente ai client di affidare la composizione e la trasmissione di una fase IVS a un servizio gestito da IVS. La composizione lato server e la trasmissione RTMP a un canale vengono richiamate tramite gli endpoint del piano di controllo IVS nella regione di origine della fase. Per ulteriori informazioni, consultare:

- Guida introduttiva allo streaming in tempo reale di IVS: abbiamo aggiunto gli endpoint SSC alla policy riportata in "Configurazione delle autorizzazioni IAM".
- Utilizzo di Amazon EventBridge con IVS Real-Time Streaming: abbiamo aggiunto nuove metriche.
- Composizione lato server: questo nuovo documento include una panoramica e istruzioni di configurazione.
- Service Quotas (streaming in tempo reale): abbiamo aggiunto nuovi limiti di frequenza delle chiamate e altre quote.
- Riferimento all'API Real-Time Streaming: abbiamo aggiunto 8 composizione ed EncoderConfiguration endpoint e 11 oggetti (ChannelDestinationConfiguration, Composition, CompositionSummary, Destination, DestinationConfiguration, DestinationSummary,, EncoderConfiguration, EncoderConfigurationSummary GridConfiguration LayoutConfiguration, e Video).

Nella Guida per l'utente dello streaming a bassa latenza di Amazon IVS, consulta:

- Abilitazione di più host su un flusso IVS: abbiamo aggiunto "Trasmissione di una fase: composizione lato client rispetto a composizione lato server" e aggiornato "4. Trasmissione della fase."

16 ottobre 2023

SDK di trasmissione Amazon IVS: Web 1.6.0 (streaming in tempo reale)

Piattaforma	Download e modifiche
SDK di trasmissione Web 1.6.0	Documentazione di riferimento: https://aws.github.io/ amazon-ivs-web-broadcast /docs/ sdk-reference • Time-To-Video (TTV) migliorato. • Configurazione `maxAudioBitrate` aggiunta, che supporta fino a 128 kbps di canali audio mono o stereo.

12 ottobre 2023

Nuove metriche e dati sui partecipanti CloudWatch

Abbiamo rilasciato le CloudWatch metriche per lo streaming in tempo reale di IVS. Consulta Monitoraggio dello streaming in tempo reale di Amazon IVS per i dettagli.

Sono stati aggiunti sei campi all'oggetto API partecipante: `browserName`, `browserVersion`, `ispName`, `osName`, `osVersion` e `sdkVersion`. Ciò influisce sulla risposta. GetParticipant Consulta la Documentazione di riferimento delle API di streaming in tempo reale IVS.

12 ottobre 2023

SDK di trasmissione Amazon IVS: Web 1.12.1 (streaming in tempo reale)

Piattaforma	Download e modifiche
SDK di trasmissione per Android 1.12.1	Documentazione di riferimento: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.1/android

Piattaforma	Download e modifiche
	• Risolto un bug in cui la chiamata `BroadcastSession.setListener` generava un errore.

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
arm64-v8a	5,853 MB	16,375 MB
armeabi-v7a	4,895 MB	10,803 MB
x86_64	6,149 MB	17,318 MB
x86	6,328 MB	17,186 MB

14 settembre 2023

SDK di trasmissione Amazon IVS: Web 1.5.2 (streaming in tempo reale)

Piattaforma	Download e modifiche
SDK di trasmissione Web 1.5.2	Documentazione di riferimento: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • È stato corretto un bug che impediva la ripubblicazione con `refreshStrategy` quando lo stato pubblicato entrava in uno stato `ERROR`ED.

23 agosto 2023

SDK di trasmissione Amazon IVS: Web 1.5.1, Android 1.12.0 e iOS 1.12.0 (streaming in tempo reale)

Piattaforma	Download e modifiche
SDK di trasmissione Web 1.5.1	Documentazione di riferimento: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • Risolto un bug con i tipi Maybe interni su 5. TypeScript • È stato aggiunto un rilevamento migliore per il supporto Simulcast. • Risolte due condizioni di competizione con `refreshStrategy` quando si prova a pubblicare. • Risolta una condizione di competizione con `refreshStrategy` quando si prova ad aggiornare i partecipanti da sottoscrivere.
Tutti i dispositivi mobili (Android e iOS)	• Risolve un problema raro in cui l'azione di pubblicazione non viene mai completata. • È stata migliorata la stabilità degli stage riducendo il numero di arresti anomali. • È stata migliorata la stabilità delle fasi risolvendo i problemi relativi alle condizioni di competizione causati dalle operazioni rapide di unione/abbandono. • Aggiunto un nuovo metodo `setOnFrameCallback` su `ImageDevice`. Ciò consente l'osservazione mentre i fotogrammi attraversano il dispositivo stesso, fornendo

Piattaforma	Download e modifiche
	informazioni sulle proporzioni delle immagini più recenti. Questo metodo può essere utilizzato anche per rilevare quando viene renderizzato il primo fotogramma per un partecipante remoto in una fase.
SDK di trasmissione per Android 1.12.0	Documentazione di riferimento: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.0/android • Android 9 ora è supportato. • Utilizzo e prestazioni della CPU migliorati.
SDK di trasmissione per iOS 1.12.0	Scarica per lo streaming in tempo reale: https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentazione di riferimento: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.0/ios • È stata corretta la firma di `IVSDeviceDiscovery.createAudioSourceWithName` in modo da restituire `IVSCustomAudioSource` invece di `IVSCustomImageSource`.

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
arm64-v8a	5,853 MB	16,375 MB
armeabi-v7a	4,895 MB	10,803 MB
x86_64	6,149 MB	17,318 MB

Architettura	Dimensione compressa	Dimensione non compressa
x86	6,328 MB	17,186 MB

Dimensione dell'SDK di trasmissione: iOS

Architettura	Dimensione compressa	Dimensione non compressa
arm64	5,06 MB	10,92 MB

7 agosto 2023

SDK di trasmissione Amazon IVS: Web 1.5.0, Android 1.11.0 e iOS 1.11.0

Piattaforma	Download e modifiche
SDK di trasmissione Web 1.5.0	Documentazione di riferimento: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • Aggiunto Simulcast: se abilitata, questa funzionalità consente all'editore di inviare livelli di video di alta e bassa qualità. Gli abbonati selezionano automaticamente la qualità ottimale in base alle condizioni della rete. Consulta Ottimizzazione dei contenuti multimediali.
Tutti i dispositivi mobili (Android e iOS)	Aggiunto Simulcast: se abilitata, questa funzione consente all'editore di inviare livelli di video di alta e bassa qualità. Gli abbonati selezionano automaticamente la qualità ottimale in base alle condizioni della rete. Consulta "Abilitazione/disabilitazione della

Piattaforma	Download e modifiche
	codifica a livelli con simulcast" nelle Guide all'SDK di trasmissione per Android e iOS.
SDK di trasmissione Android 1.11.0	Documentazione di riferimento: https://aws.github.io/ /1.11.0/android amazon-ivs-broadcast-docs • È stato risolto un problema per cui la creazione di più fasi alla fine causava un arresto anomalo. (Il numero esatto di fasi dipende dal dispositivo.)
SDK di trasmissione iOS 1.11.0	Scarica per lo streaming in tempo reale: https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip Documentazione di riferimento: https://aws.github.io/ /1.11.0/ios amazon-ivs-broadcast-docs • È stata corretta la firma di `IVSDeviceDiscovery.createAudioSourceWithName` in modo da restituire `IVSCustomAudioSource` invece di `IVSCustomImageSource`.

Dimensione dell'SDK di trasmissione: Android

Architettura	Dimensione compressa	Dimensione non compressa
arm64-v8a	5,811 MB	16,186 MB
armeabi-v7a	4,857 MB	10,646 MB
x86_64	6,108 MB	17,122 MB

Architettura	Dimensione compressa	Dimensione non compressa
x86	6,289 MB	16,994 MB

Dimensione dell'SDK di trasmissione: iOS

Architettura	Dimensione compressa	Dimensione non compressa
arm64	5,030 MB	10,810 MB

7 agosto 2023

Streaming in tempo reale

Lo streaming in tempo reale di Amazon Interactive Video Service (IVS) ti consente di fornire streaming live con una latenza che può essere inferiore a 300 millisecondi dall'host allo spettatore.

Questa release comprende le principali modifiche alla documentazione. La pagina iniziale della documentazione IVS ora dispone di sezioni separate per lo streaming in tempo reale e lo streaming a bassa latenza. Ogni sezione ha una propria Guida per l'utente e la Documentazione di riferimento delle API. Per i dettagli della documentazione, consulta la Cronologia dei documenti (sia per il tempo reale che per la bassa latenza). Per lo streaming in tempo reale, inizia con Guida per l'utente dello streaming in tempo reale IVS e Documentazione di riferimento delle API di streaming in tempo reale IVS.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.