Benutzerhandbuch für Echtzeit-Streaming

# Amazon IVS

# Amazon IVS: Benutzerhandbuch für Echtzeit-Streaming

# Table of Contents

# Was ist Amazon-IVS-Echtzeit-Streaming?

Amazon Interactive Video Service (IVS)-Echtzeit-Streaming bietet Ihnen alles, was Sie benötigen, um Ihren Anwendungen Audio und Video in Echtzeit hinzuzufügen.

Stärken:

- Latenz in Echtzeit – Entwickeln Sie Anwendungen für latenzsensitive Anwendungsfälle und helfen Sie Ihren Zuschauern, mit IVS-Echtzeit-Streaming in Verbindung zu bleiben und zu interagieren. Stellen Sie Live-Streams mit einer Latenz bereit, die vom Host bis zum Zuschauer unter 300 Millisekunden liegen kann.
- Hohe Parallelität – Nutzen Sie das Potenzial groß angelegter Interaktionen mit IVS-Echtzeit-Streaming. Bietet Platz für ein Publikum von bis zu 10 000 Zuschauern und ermöglicht es bis zu 12 Moderatoren, die virtuelle Stage zu betreten.
- Für Mobilgeräte optimiert – IVS-Echtzeit-Streaming ist für mobile Anwendungsfälle optimiert und eignet sich für eine Vielzahl von Geräten und Netzwerkfunktionen. Durch die Integration der Amazon IVS Broadcast SDKs für Android und iOS können Ihre Benutzer als Hosts oder Zuschauer interagieren und hochwertige Live-Streams auf ihren Mobilgeräten genießen.

Anwendungsfälle:

- Gastspots – Entwickeln Sie Anwendungen, mit denen Gastgeber Gäste „auf der Stage“ bewerben können, sodass Zuschauer zu Hosts für Interaktionen in Echtzeit werden.
- Versus-Modus (VS) – Produzieren Sie Erlebnisse mit nebeneinanderliegenden Wettbewerben und lassen Sie die Zuschauer zuschauen, wenn die Hosts in Echtzeit gegeneinander antreten.
- Audioräume – Laden Sie Zuhörer ein, als Gäste an der Konversation teilzunehmen, und fördern Sie ein intensiveres Engagement in Ihren Audioräumen.
- Live-Videoauktionen – Verwandeln Sie Auktionen in interaktive Videoevents und sorgen Sie mit Latenz in Echtzeit für Spannung und Integrität.

Neben der Produktdokumentation hier finden Sie https://ivs.rocks/, eine spezielle Website zum Durchsuchen veröffentlichter Inhalte (Demos, Codebeispiele, Blog-Posts), Kostenschätzungen und Erleben von Amazon IVS durch Live-Demos.

# Globale Lösung, regionale Kontrolle

## Streaming und Anzeigen sind global

Sie können Amazon IVS verwenden, um für Zuschauer weltweit zu streamen:

- Wenn Sie streamen, nimmt Amazon IVS automatisch Videos an einem Standort in Ihrer Nähe auf.
- Zuschauer können Ihre Livestreams weltweit ansehen.

Eine andere Möglichkeit, dies zu sagen, ist, dass die „Datenebene“ global ist. Die Datenebene bezieht sich auf Streaming/Aufnahme und Betrachtung.

## Kontrolle ist Regional

Während die Amazon IVS-Datenebene global ist, ist die „Steuerungsebene“ regional. Die Steuerebene bezieht sich auf die Amazon-IVS-Konsole, API und Ressourcen (Stages).

Man könnte auch sagen, dass Amazon IVS ein „regionaler AWS-Service“ ist. Das heißt, Amazon IVS-Ressourcen sind in jeder Region unabhängig von ähnlichen Ressourcen in anderen Regionen. Beispielsweise ist ein Stage, den Sie in einer Region erstellen, unabhängig von Stages, die Sie in anderen Regionen erstellen.

Wenn Sie Ressourcen verwenden (z. B. einen Kanal erstellen), müssen Sie die Region angeben, in der er erstellt wird. Wenn Sie anschließend Ressourcen verwalten, müssen Sie dies von demselben Bereich aus tun, in dem sie erstellt wurden.

| Bei Verwendung der ... | Sie geben die Region an, indem Sie... |
|---|---|
| Amazon IVS-Konsole | Verwendung von Auswählen einer Region oben rechts in der Navigatio nsleiste. |
| Amazon IVS-API | Verwenden des entsprechenden Service-Endpunktes. Sehen Sie die API-Referenz zu Amazon-IVS-Echtzeit-Streaming.<br><br>(Wenn Sie über ein SDK auf die API zugreifen, richten Sie den `region`-Parameter des SDKs ein. Siehe Tools zum Entwickeln in AWS.) |

| Bei Verwendung der ... | Sie geben die Region an, indem Sie... |
| --- | --- |
| AWS CLI | Entweder:<br>• Anhängen von `--region <aws-region>` an Ihren CLI-Befehl.<br>• Platzieren Sie die Region in Ihre lokale AWS-Konfigurationsdatei. |

Denken Sie daran, dass Sie unabhängig von der Region, in der ein Stage erstellt wurde, von überall zu Amazon IVS streamen können und Zuschauer es von überall aus ansehen können.

# Erste Schritte mit IVS-Echtzeit-Streaming

Dieses Dokument führt Sie durch die Schritte zur Integration von Amazon-IVS-Echtzeit-Streaming in Ihre Anwendung.

Themen



## Einführung

### Voraussetzungen

Bevor Sie Echtzeit-Streaming zum ersten Mal verwenden können, müssen Sie die folgenden Aufgaben erledigen. Anleitungen finden Sie unter Erste Schritte mit IVS-Streaming mit niedriger Latenz.

- Erstellen eines AWS-Kontos
- Richten Sie Root-Benutzer und Administratoren ein.

### Andere Referenzen:

- Referenz zum IVS-Web-Broadcast-SDK
- Referenz zum IVS-Android-Broadcast-SDK
- Referenz zum IVS-iOS-Broadcast-SDK
- Referenz zur API von IVS-Echtzeit-Streaming

# Terminologie für Echtzeit-Streaming

| Begriff | Beschreibung |
|---|---|
| Stufe | Ein virtueller Raum, in dem die Teilnehmer Videos in Echtzeit austauschen können. |
| Host | Ein Teilnehmer, der ein lokales Video auf die Stage sendet. |
| Zuschauer | Ein Teilnehmer, der ein Video der Hosts erhält. |
| Teilnehmer | Ein Benutzer, der als Host oder Zuschauer mit der Stage verbunden ist. |
| Teilnehmer-Token | Ein Token, das einen Teilnehmer authentifiziert, wenn er einer Stage beitritt. |
| Broadcast-SDK | Eine Clientbibliothek, die es den Teilnehmern ermöglicht, Videos zu senden und zu empfangen. |

## Übersicht über die Schritte

1. the section called “IAM-Berechtigungen einrichten” – Erstellen Sie eine AWS-Richtlinie für Identity and Access Management (IAM), die Benutzern grundlegende Berechtigungen gewährt, und weisen Sie diese Richtlinie Benutzern zu.
2. Schaffen Sie eine Stage – Schaffen Sie eine virtuelle Umgebung, in der die Teilnehmer Videos in Echtzeit austauschen können.
3. Verteilen Sie Teilnehmer-Token – Senden Sie Tokens an die Teilnehmer, damit sie Ihrer Stage beitreten können.

4. Integrieren Sie das IVS-Broadcast-SDK – Fügen Sie das Broadcast-SDK zu Ihrer Anwendung hinzu, damit die Teilnehmer Videos senden und empfangen können: the section called “Web”, the section called “Android” und the section called “iOS”.
5. Video veröffentlichen und abonnieren – Senden Sie Ihr Video an die Stage und erhalten Sie Videos von anderen Hosts: the section called “Web”, the section called “Android”, und the section called “iOS”.

# IAM-Berechtigungen einrichten

Als Nächstes müssen Sie eine AWS Identity and Access Management (IAM)-Richtlinie erstellen, die Benutzern einen grundlegenden Satz an Berechtigungen gewährt (z. B. zum Erstellen einer Amazon IVS-Stufe und zum Erstellen von Teilnehmer-Tokens) und diese Richtlinie den Benutzern zuweisen. Die Berechtigungen können Sie beim Erstellen eines neuen Benutzers zuweisen oder einem vorhandenen Benutzer hinzufügen. Im Folgenden sind beide Verfahren angegeben.

Weitere Informationen (z. B. Informationen zu IAM-Benutzern und -Richtlinien, zum Anhängen einer Richtlinie an einen Benutzer und zum Beschränken der Möglichkeiten von Benutzern mit Amazon IVS) finden Sie unter:

- Erstellen eines IAM-Benutzers im IAM-Benutzerhandbuch
- Die Informationen in Amazon-IVS-Sicherheit zu IAM und „Verwaltete Richtlinien für IVS“.
- Die IAM-Informationen in Amazon-IVS-Sicherheit

Sie können entweder eine vorhandene von AWS verwaltete Richtlinie für Amazon IVS verwenden oder eine neue Richtlinie erstellen, die die Berechtigungen anpasst, die Sie einer Reihe von Benutzern, Gruppen oder Rollen gewähren möchten. Beide Vorgehensweisen werden nachfolgend beschrieben.

## Verwenden einer vorhandenen Richtlinie für IVS-Berechtigungen

In den meisten Fällen möchten Sie für Amazon IVS eine von AWS verwaltete Richtlinie verwenden. Diese werden ausführlich im Abschnitt Verwaltete Richtlinien für IVS von IVS-Sicherheit beschrieben.

- Verwenden Sie die von AWS verwaltete Richtlinie `IVSReadOnlyAccess`, um Ihren Anwendungsentwicklern Zugriff auf alle IVS-Get- und List-API-Endpunkte zu gewähren (sowohl für Streaming mit niedriger Latenz als auch für Echtzeit-Streaming).

- Verwenden Sie die von AWS verwaltete Richtlinie `IVSFullAccess`, um Ihren Anwendungsentwicklern Zugriff auf alle IVS-API-Endpunkte zu gewähren (sowohl für Streaming mit niedriger Latenz als auch für Echtzeit-Streaming).

## Optional: Eine benutzerdefinierte Richtlinie für Amazon-IVS-Berechtigungen erstellen

Dazu gehen Sie wie folgt vor:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich Policies und dann Create policy. Das Fenster Berechtigungen angeben wird geöffnet.
3. Wählen Sie im Fenster Berechtigungen angeben die Registerkarte JSON. Kopieren Sie die folgende IVS-Richtlinie und fügen Sie sie in den Textbereich Richtlinien-Editor ein. (Die Richtlinie enthält nicht alle Amazon IVS-Aktionen. Sie können je nach Anforderung Zugriffsberechtigungen für Endpunkte hinzufügen/löschen (Zulassen/Verweigern). Einzelheiten zu den IVS-Endpunkten finden Sie unter API-Referenz zum IVS-Echtzeit-Streaming.)

```
{
   "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. Wählen Sie im Fenster Berechtigungen angeben die Option Weiter aus (scrollen Sie zum unteren Ende des Fensters, um diese Option anzuzeigen). Das Fenster Überprüfen und erstellen wird geöffnet.
5. Geben Sie im Fenster Überprüfen und erstellen einen Richtliniennamen ein und fügen Sie optional eine Beschreibung hinzu. Notieren Sie sich den Namen der Richtlinie, da Sie ihn beim Erstellen von Benutzern (weiter unten) benötigen. Wählen Sie Create policy (Richtlinie erstellen) aus (am unteren Ende des Fensters).
6. Sie gelangen zurück zum IAM-Konsolenfenster, in dem per Banner bestätigt werden sollte, dass die neue Richtlinie erstellt wurde.

## Erstellen Sie einen neuen Benutzer und fügen Sie Berechtigungen hinzu

### IAM-Benutzer-Zugriffsschlüssel

IAM-Zugriffsschlüssel bestehen aus einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel. Sie dienen zum Signieren Ihrer programmgesteuerten Anforderungen an AWS. Wenn Sie noch keine Zugriffsschlüssel besitzen, können Sie diese über die AWS-Managementkonsole erstellen. Verwenden Sie als bewährte Methode keine Zugriffsschlüssel für Root-Benutzer.

Einen geheimen Zugriffsschlüssel können Sie nur beim Erstellen von Zugriffsschlüsseln anzeigen oder herunterladen. Später kann er nicht mehr wiederhergestellt werden. Sie können jedoch jederzeit neue Zugriffsschlüssel erstellen. Dazu benötigen Sie die Berechtigungen zum Ausführen der erforderlichen IAM-Aktionen.

Bewahren Sie Zugriffsschlüssel stets sicher auf. Geben Sie sie niemals an Dritte weiter (selbst wenn eine Anfrage von Amazon zu stammen scheint). Weitere Informationen finden Sie unter Verwalten der Zugriffsschlüssel für IAM-Benutzer im -IAM-Benutzerhandbuch.

## Verfahren

Dazu gehen Sie wie folgt vor:

1. Wählen Sie im Navigationsbereich die Option Benutzer und dann Benutzer erstellen. Das Fenster Benutzerdetails angeben wird geöffnet.
2. Gehen Sie im Fenster Benutzerdetails angeben wie folgt vor:
   a. Geben Sie unter Benutzerdetails den neuen Benutzernamen ein, der erstellt werden soll.
   b. Aktivieren Sie Benutzerzugriff auf AWS-Managementkonsole bereitstellen.
   c. Wählen Sie unter Console password (Konsolenpasswort) Autogenerated password (Automatisch generiertes Passwort).
   d. Aktivieren Sie Benutzer muss bei der nächsten Anmeldung ein neues Passwort erstellen.
   e. Wählen Sie Weiter aus. Das Fenster Berechtigungen festlegen wird geöffnet.
3. Wählen Sie unter Berechtigungen festlegen die Option Richtlinien direkt zuweisen aus. Das Fenster Berechtigungsrichtlinien wird geöffnet.
4. Geben Sie im Suchfeld einen IVS-Richtliniennamen ein (entweder eine von AWS verwaltete Richtlinie oder Ihre zuvor erstellte benutzerdefinierte Richtlinie). Wenn sie gefunden wurde, aktivieren Sie das Kästchen, um die Richtlinie auszuwählen.
5. Wählen Sie Weiter (unten im Fenster). Das Fenster Überprüfen und erstellen wird geöffnet.
6. Vergewissern Sie sich im Fenster Überprüfen und erstellen, ob alle Benutzerdetails korrekt sind, und wählen Sie dann Benutzer erstellen aus (unten im Fenster).
7. Das Fenster Passwort abrufen wird geöffnet, das Ihre Details zur Anmeldung an der Konsole enthält. Bewahren Sie diese Informationen sicher auf, damit Sie in Zukunft darauf zurückgreifen können. Klicken Sie abschließend auf Zurück zur Benutzerliste.

## Hinzufügen von Berechtigungen zu einem vorhandenen Benutzer

Dazu gehen Sie wie folgt vor:

1. Melden Sie sich bei der AWS-Managementkonsole an und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie im Navigationsbereich Benutzer aus und wählen Sie dann einen vorhandenen Benutzernamen aus, der aktualisiert werden soll. (Wählen Sie den Namen aus, indem Sie darauf klicken. Aktivieren Sie nicht das Auswahlfeld.)
3. Wählen Sie auf der Seite Zusammenfassung in der Registerkarte Berechtigungen die Option Berechtigungen hinzufügen aus. Das Fenster Berechtigungen hinzufügen wird geöffnet.
4. Wählen Sie die Option Attach existing policies directly (Vorhandene Richtlinien direkt anfügen) aus. Das Fenster Berechtigungsrichtlinien wird geöffnet.
5. Geben Sie im Suchfeld einen IVS-Richtliniennamen ein (entweder eine von AWS verwaltete Richtlinie oder Ihre zuvor erstellte benutzerdefinierte Richtlinie). Wenn die Richtlinie gefunden wurde, aktivieren Sie das Kästchen, um die Richtlinie auszuwählen.
6. Wählen Sie Weiter (unten im Fenster). Das Fenster Überprüfung wird geöffnet.
7. Wählen Sie im Fenster Überprüfung die Option Berechtigungen hinzufügen aus (unten im Fenster).
8. Vergewissern Sie sich auf der Seite Summary (Zusammenfassung), dass die IVS-Richtlinie hinzugefügt wurde.

# Erstellen einer Bühne

Bei einer Stage handelt es sich um eine virtuellen Umgebung, in der die Teilnehmer Audio und Video in Echtzeit austauschen können. Sie ist die grundlegende Ressource der Echtzeit-Streaming-API. Sie können eine Stufe entweder über die Konsole oder den CreateStage Endpunkt erstellen.

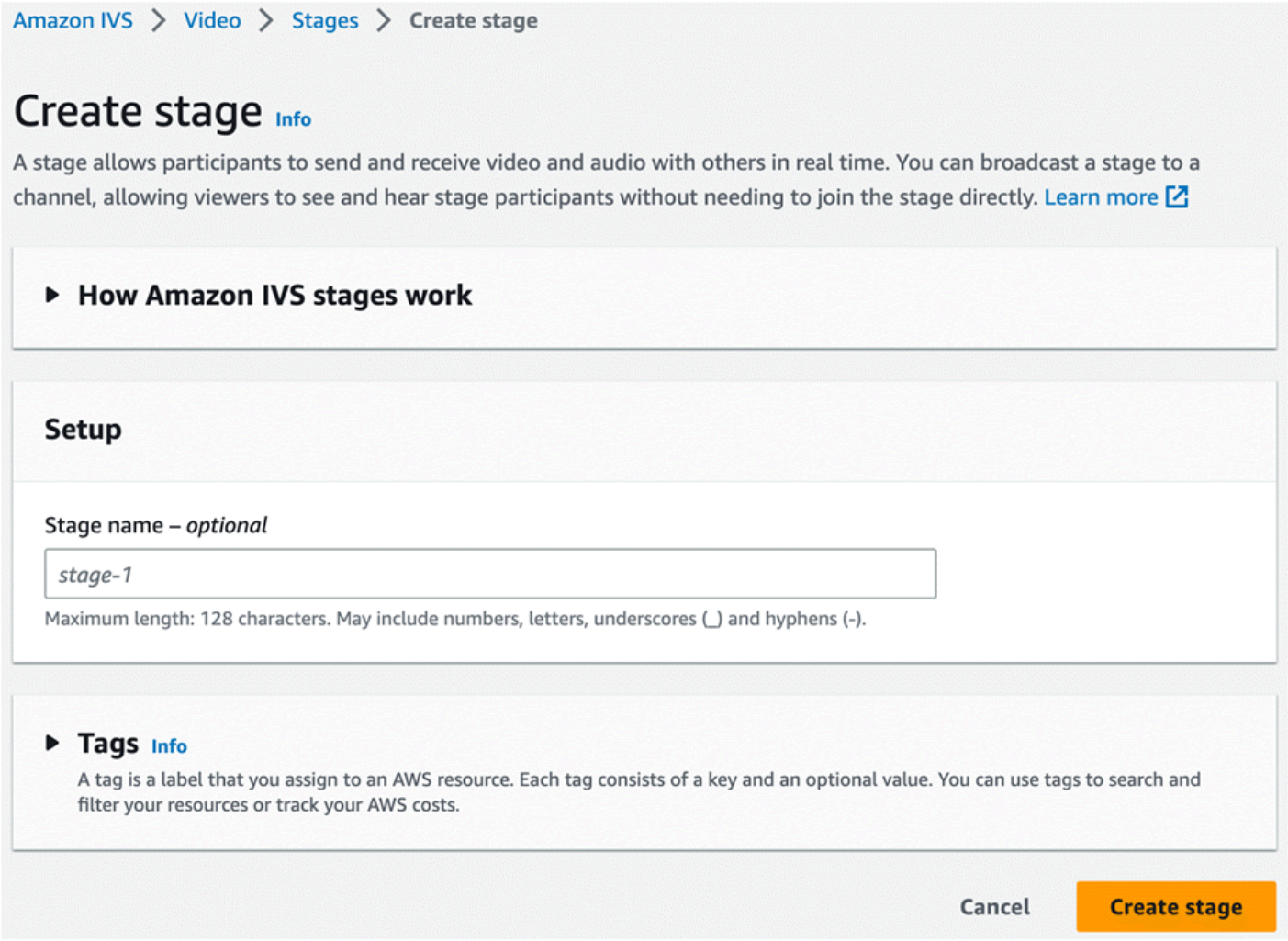
Wir empfehlen, dass Sie nach Möglichkeit für jede logische Sitzung eine neue Stage erstellen und diese löschen, wenn Sie fertig sind, anstatt alte Stages für eine mögliche Wiederverwendung beizubehalten. Wenn veraltete Ressourcen (alte Stages, die nicht wiederverwendet werden sollen) nicht bereinigt werden, erreichen Sie wahrscheinlich schneller das Limit der maximalen Anzahl an Stages.

## Anleitung für die Konsole

1. Öffnen Sie die Amazon-IVS-Konsole.

   (Sie können auf die Amazon-IVS-Konsole auch über die AWS-Managementkonsole zugreifen.)

2. Wählen Sie im linken Navigationsbereich Bühnen und dann Bühne erstellen aus. Das Fenster Bühne erstellen wird angezeigt.




3. Geben Sie optional einen Bühnennamen ein. Wählen Sie Bühne erstellen aus, um die Bühne zu erstellen. Die Seite mit den Bühnendetails für die neue Bühne wird angezeigt.

## CLI-Anweisungen

Informationen zum Installieren der AWS-CLI finden Sie unter Installieren oder Aktualisieren der neuesten Version der AWS-CLI.

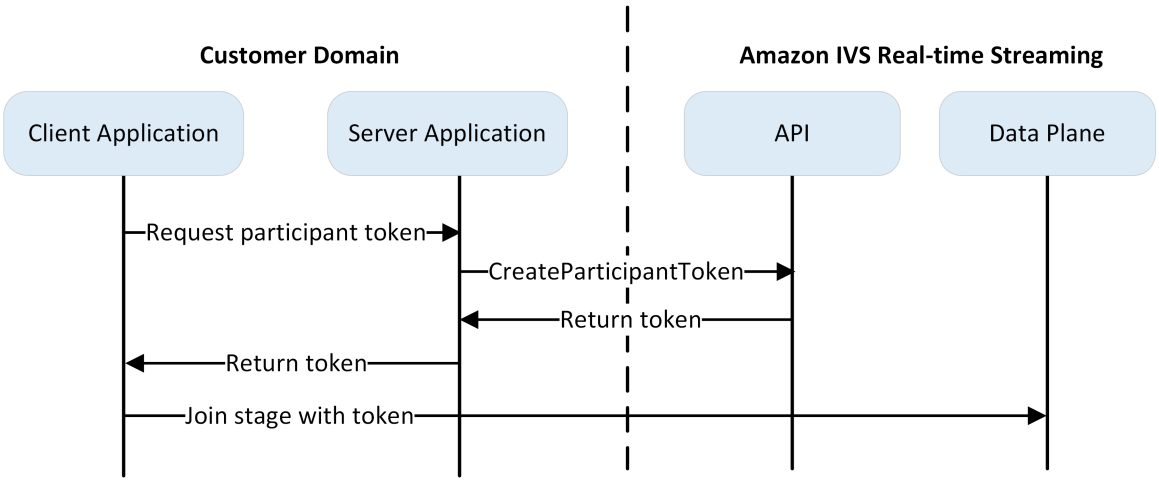
Nun können Sie mit der CLI Ressourcen erstellen und verwalten. Die Bühnen-API befindet sich unter dem Namespace ivs-realtime. Beispiel zum Erstellen einer Bühne:

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

Die Antwort ist:

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

# Verteilen von Teilnehmertoken




Jetzt, da Sie eine Stage haben, müssen Sie Tokens erstellen und an die Teilnehmer verteilen, damit sie der Stage beitreten und mit dem Senden und Empfangen von Videos beginnen können.

Wie oben gezeigt, fragt eine Clientanwendung Ihre Serveranwendung nach einem Token, und die Serveranwendung ruft CreateParticipantToken mithilfe eines AWS SDK oder einer SigV4-signierten Anforderung auf. Da AWS-Anmeldeinformationen zum Aufrufen der API verwendet werden, sollte das Token in einer sicheren serverseitigen Anwendung generiert werden, nicht in der clientseitigen Anwendung.

Beim Erstellen eines Teilnehmer-Tokens können Sie optional die Funktionen angeben, die durch dieses Token aktiviert werden. Die Standardeinstellung ist `PUBLISH` und `SUBSCRIBE`, was es dem

Teilnehmer ermöglicht, Audio und Video zu senden und zu empfangen, aber Sie können Tokens mit einer Teilmenge von Funktionen ausgeben. Sie könnten zum Beispiel einen Token ausgeben, der nur die Fähigkeit `SUBSCRIBE` für Moderatoren enthält. In diesem Fall könnten die Moderatoren die Teilnehmer sehen, die ein Video senden, aber kein eigenes Video senden.

Sie können Teilnehmer-Token über die Konsole oder CLI zu Test- und Entwicklungszwecken erstellen. Höchstwahrscheinlich möchten Sie sie jedoch mit dem AWS-SDK in Ihrer Produktionsumgebung erstellen.

Sie benötigen eine Möglichkeit, um Token von Ihrem Server an alle Clients zu verteilen (z. B. über eine API-Anforderung). Diese Funktionalität wird von uns nicht bereitgestellt. Für diese Anleitung können Sie die Token einfach kopieren und in den folgenden Schritten in den Client-Code einfügen.

Wichtig: Behandeln Sie Token als nicht transparent, d. h. entwickeln Sie keine Funktionen, die auf Tokeninhalten basieren. Das Format von Token könnte sich in Zukunft ändern.

## Anleitung für die Konsole

1. Navigieren Sie zu der Phase, die Sie im vorherigen Schritt erstellt haben.
2. Wählen Sie Teilnehmer-Token erstellen aus. Das Fenster Teilnehmer-Token erstellen erscheint.
3. Geben Sie eine Benutzer-ID ein, die dem Token zugeordnet werden soll. Dies kann jeder UTF-8-kodierte Text sein.
4. Wählen Sie Teilnehmer-Token erstellen aus.
5. Kopieren Sie das Token. Wichtig: Achten Sie darauf, das Token zu speichern. IVS speichert es nicht und Sie können es später nicht abrufen.

## CLI-Anweisungen

Das Erstellen eines Chat-Tokens mit der AWS CLI ist eine erweiterte Option und erfordert, dass Sie zuerst die CLI auf Ihrem Computer herunterladen und konfigurieren. Informationen zu den ersten Schritten finden Sie im Benutzerhandbuch für die AWS-Befehlszeilenschnittstelle. Beachten Sie, dass die Generierung von Token mit der AWS-CLI für Testzwecke gut geeignet ist. Für den produktiven Einsatz empfehlen wir jedoch, Token auf der Serverseite mit dem AWS-SDK zu generieren (siehe Anweisungen unten).

1. Führen Sie den `create-participant-token`-Befehl mit dem Stage-ARN aus. Fügen Sie eine der folgenden Funktionen ein: `"PUBLISH"`, `"SUBSCRIBE"`.

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

2. Dies gibt ein Teilnehmer-Token zurück:

```
{
    "participantToken": {
        "capabilities": [
            "PUBLISH",
            "SUBSCRIBE"
        ],
        "expirationTime": "2023-06-03T07:04:31+00:00",
        "participantId": "tU06DT5jCJeb",
        "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtdmVFTm9sUyIsInJl
TaKjllW9Qac6c5xBrdAk"    }
}
```

3. Speichern Sie dieses Token. Sie benötigen dies, um der Stage beizutreten und Videos zu senden und zu empfangen.

## AWS-SDK-Anweisungen

Sie können das AWS-SDK für die Erstellung von Tokens verwenden. Im Folgenden finden Sie Anweisungen für das AWS SDK mit JavaScript.

Wichtig: Dieser Code muss serverseitig ausgeführt und seine Ausgabe an den Client übergeben werden.

Voraussetzung: Um das folgende Codebeispiel zu verwenden, müssen Sie aws-sdk/client-ivs-realtime package installieren. Weitere Informationen finden Sie unter Erste Schritte mit dem AWS SDK for JavaScript.

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

const ivsRealtimeClient = new IVSRealTimeClient({ region: 'us-west-2' });
const stageArn = 'arn:aws:ivs:us-west-2:123456789012:stage/L210UYabcdef';
const createStageTokenRequest = new CreateParticipantTokenCommand({
  stageArn,
});
```

```
const response = await ivsRealtimeClient.send(createStageTokenRequest);
console.log('token', response.participantToken.token);
```

# Integrieren Sie das IVS-Broadcast-SDK

IVS bietet ein Broadcast-SDK für Web, Android und iOS, das Sie in Ihre Anwendung integrieren können. Das Broadcast-SDK wird sowohl zum Senden als auch zum Empfangen von Videos verwendet. In diesem Abschnitt schreiben wir eine einfache Anwendung, mit der zwei oder mehr Teilnehmer in Echtzeit interagieren können. Die folgenden Schritte führen Sie durch das Erstellen einer App namens BasicRealTime. Der vollständige App-Code ist auf CodePen und GitHub:

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

## Web

### Dateien einrichten

Richten Sie zunächst Ihre Dateien ein, indem Sie einen Ordner und eine erste HTML- und JS-Datei erstellen:

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

Sie können das Broadcast-SDK mit einem Script-Tag oder npm installieren. Unser Beispiel verwendet der Einfachheit halber das Script-Tag, kann aber leicht geändert werden, wenn Sie npm später verwenden möchten.

### Verwenden eines Skript-Tags

Das Web-Broadcast-SDK wird als JavaScript Bibliothek verteilt und kann unter https://web-broadcast.live-video.net/1.8.0/amazon-ivs-web-broadcast.js abgerufen werden.

Wenn sie per `<script>`-Tag geladen wird, stellt die Bibliothek eine globale Variable im Fensterbereich namens `IVSBroadcastClient` bereit.

### Verwenden von npm

So installieren Sie das npm-Paket:

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

Sie können jetzt auf das IVS-BroadcastClient Objekt zugreifen:

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

## Android

### Erstellen Sie das Android-Projekt

1. Erstellen Sie ein Neues Projekt mit Android Studio.
2. Wählen Sie Aktivität „Leere Ansichten“.

   Hinweis: In einigen älteren Versionen von Android Studio heißt die ansichtsbasierte Aktivität Leere Aktivität. Wenn Ihr Android-Studio-Fenster Leere Aktivität und nicht Leere Ansichten-Aktivität anzeigt, wählen Sie Leere Aktivität. Andernfalls wählen Sie nicht Leere Aktivität, da wir View-APIs verwenden werden (nicht Jetpack Compose).
3. Geben Sie Ihrem Projekt einen Namen, wählen Sie dann Fertig.

### Installieren Sie das Broadcast-SDK

Wenn Sie der Android-Entwicklungsumgebung die Amazon-IVS-Android-Broadcast-Bibliothek hinzufügen möchten, fügen Sie die Bibliothek der `build.gradle` – wie hier gezeigt – (für die neueste Version des Amazon IVS Broadcast SDK) zu Ihren Modulen hinzu. In neueren Projekten ist `mavenCentral` das Repository ist möglicherweise bereits in Ihrer `settings.gradle`-Datei. Wenn das der Fall ist, können Sie den `repositories`-Block weglassen. Für unser Beispiel müssen wir auch die Datenbindung im Block `android` aktivieren.

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
```

```
}

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

Um das SDK manuell zu installieren, laden Sie alternativ die neueste Version von diesem Speicherort herunter:

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

# iOS

## Erstellen des iOS-Projekts

1. Erstellen eines neuen Xcode-Projekts.
2. Für Plattform wählen Sie iOS.
3. Für Anwendung wählen Sie App.
4. Geben Sie den Namen des Produkts Ihrer Anwendung ein und wählen Sie Weiter.
5. Wählen (navigieren Sie zu) einem Verzeichnis, in dem das Projekt gespeichert werden soll, und wählen Sie dann Erstellen.

Als Nächstes müssen Sie das SDK einbringen. Wir empfehlen Ihnen, das Broadcast-SDK über zu integrieren CocoaPods. Alternativ können Sie das Framework manuell zu Ihrem Projekt hinzufügen. Beide Methoden werden im Folgenden beschrieben.

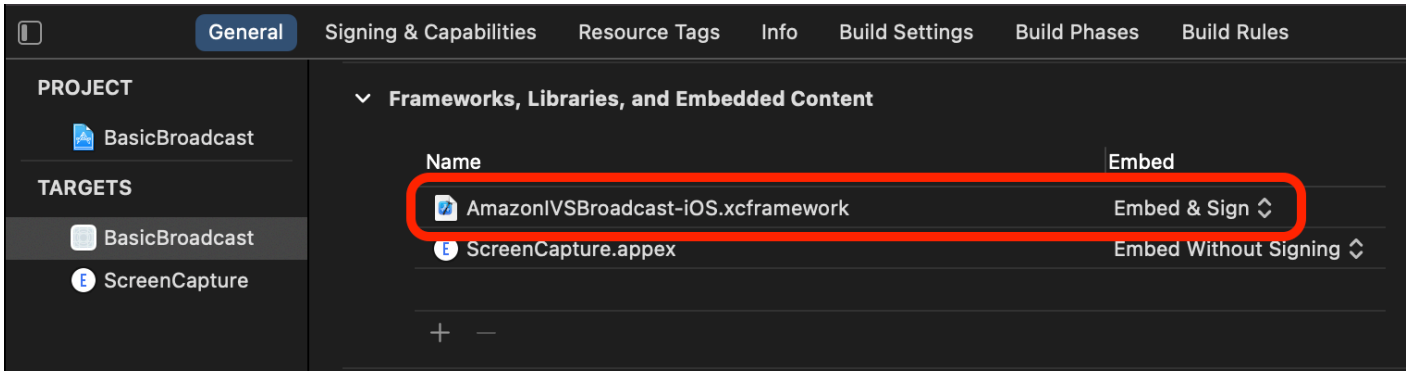
## Empfohlen: Installieren des Broadcast SDK (CocoaPods)

Angenommen, Ihr Projektname lautet `BasicRealTime`, erstellen Sie ein `Podfile` im Projektordner mit dem folgenden Inhalt und führen dann `pod install` aus:

```
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
```

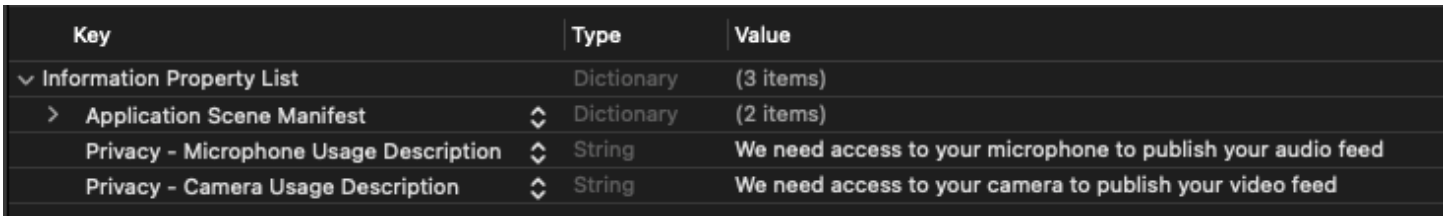
### Manuelles Installieren der Framework

1. Laden Sie die neueste Version von https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip herunter.
2. Extrahieren Sie den Inhalt des Archivs. `AmazonIVSBroadcast.xcframework` enthält das SDK für Gerät und Simulator.
3. Betten Sie `AmazonIVSBroadcast.xcframework` ein, indem Sie es in den Abschnitt Rahmenbedingungen, Bibliotheken und eingebettete Inhalte auf der Registerkarte Allgemein für Ihr Anwendungsziel ziehen:




### Konfigurieren der -Berechtigungen

Sie müssen die `Info.plist` Ihres Projekts aktualisieren, um zwei neue Einträge hinzuzufügen für `NSCameraUsageDescription` und `NSMicrophoneUsageDescription`. Geben Sie für die Werte benutzerfreundliche Erklärungen an, warum Ihre Anwendung nach Kamera- und Mikrofonzugriff fragt.




## Video veröffentlichen und abonnieren

Weitere Informationen finden Sie unten für Web , Android und iOS .

# Web

## HTML-Boilerplate erstellen

Lassen Sie uns zunächst das HTML-Boilerplate erstellen und die Bibliothek als Script-Tag importieren:

```
<!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>
```

## Token-Eingabe akzeptieren und Schaltflächen zum Beitritten/Verlassen hinzufügen

Hier füllen wir den Hauptteil mit unseren Eingabekontrollen aus. Diese nehmen das Token als Eingabe und richten Schaltflächen für Beitreten und Verlassen ein. Normalerweise fordern Anwendungen das Token von der API Ihrer Anwendung an, aber in diesem Beispiel kopieren Sie das Token und fügen es in die Token-Eingabe ein.

```
<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>
```

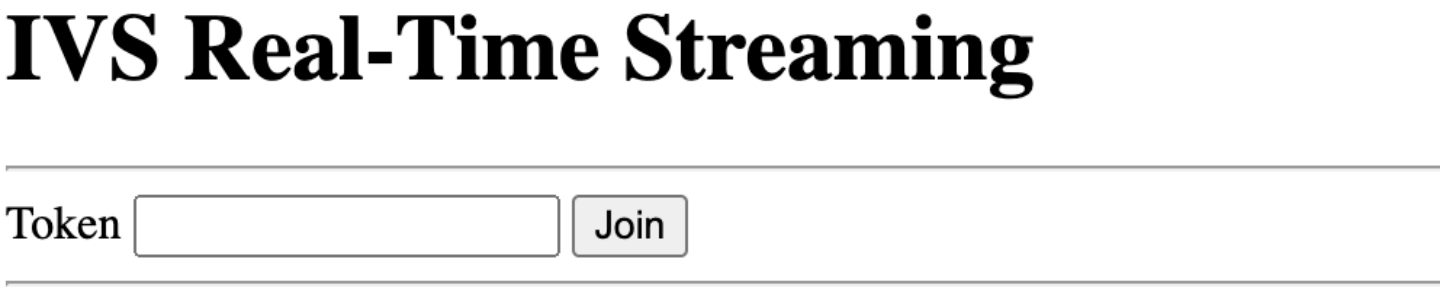
```
<button class="button" id="leave-button" style="display: none;">Leave</button>
<hr />
```

## Mediencontainer-Elemente hinzufügen

Diese Elemente werden die Medien für unsere lokalen und externen Teilnehmer bereitstellen. Wir fügen ein Script-Tag hinzu, um die in `app.js` definierte Logik unserer Anwendung zu laden.

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

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

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

Damit ist die HTML-Seite fertig. Sie sollten sie sehen, wenn Sie `index.html` in einem Browser laden:




## Erstellen von app.js

Gehen wir zur Definition des Inhalts unserer `app.js`-Datei. Importieren Sie zunächst alle erforderlichen Eigenschaften aus der globalen Version des SDK:

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

## Anwendungsvariablen erstellen

Erstellen Sie Variablen, um Verweise auf unsere HTML-Elemente für die Schaltflächen Beitreten und Verlassen zu speichern und den Status für die Anwendung zu speichern:

```
let joinButton = document.getElementById("join-button");
let leaveButton = document.getElementById("leave-button");

// Stage management
let stage;
let joining = false;
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
let micStageStream;
```

## JoinStage 1 erstellen: Definieren Sie die Funktion und validieren Sie die Eingabe

Die Funktion `joinStage` nimmt das Eingabe-Token, stellt eine Verbindung zur Stage her und beginnt mit der Veröffentlichung von Video- und Audiodaten, die von `getUserMedia` empfangen werden.

Zu Beginn definieren wir die Funktion und validieren den Status und die Token-Eingabe. Wir werden diese Funktion in den nächsten Abschnitten näher erläutern.

```
const joinStage = async () => {
  if (connected || joining) {
    return;
  }
  joining = true;

  const token = document.getElementById("token").value;

  if (!token) {
    window.alert("Please enter a participant token");
    joining = false;
    return;
  }

  // Fill in with the next sections
};
```

## JoinStage 2 erstellen: Medien zum Veröffentlichen abrufen

Hier sind die Medien, die auf der Stage veröffentlicht werden:

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

async function getMic() {
  return navigator.mediaDevices.getUserMedia({
    video: false,
    audio: true
  });
}

// Retrieve the User Media currently set on the page
localCamera = await getCamera();
localMic = await getMic();

// Create StageStreams for Audio and Video
cameraStageStream = new LocalStageStream(localCamera.getVideoTracks()[0]);
micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);
```

## JoinStage 3 erstellen: Definieren Sie die Stagestrategie und erstellen Sie die Stage

Diese Stagestrategie ist das Herzstück der Entscheidungslogik, anhand derer das SDK entscheidet, was veröffentlicht und welche Teilnehmer abonniert werden sollen. Weitere Informationen zum Zweck der Funktion finden Sie unter Strategie.

Diese Strategie ist einfach. Nachdem Sie die Stage betreten haben, veröffentlichen Sie die soeben abgerufenen Streams und abonnieren die Audio- und Videodaten aller Remote-Teilnehmer:

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

```
  shouldSubscribeToParticipant() {
    return SubscribeType.AUDIO_VIDEO;
  }
};

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

## JoinStage 4 erstellen: Stageereignisse verarbeiten und Medien rendern

Stages geben viele Ereignisse ab. Wir müssen auf die `STAGE_PARTICIPANT_STREAMS_ADDED` und `STAGE_PARTICIPANT_LEFT` hören, um Medien auf und von der Seite zu rendern und zu entfernen. Eine umfassendere Reihe von Ereignissen finden Sie unter Ereignisse.

Beachten Sie, dass wir hier vier Hilfsfunktionen erstellen, die uns bei der Verwaltung der erforderlichen DOM-Elemente unterstützen:`setupParticipant`, `teardownParticipant`, `createVideoEl` und `createContainer`.

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

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

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

    let streamsToDisplay = streams;

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

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

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


// Helper functions for managing DOM

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

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

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

  return videoEl;
}

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

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

function createVideoEl(id) {
```

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

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

  return participantContainer;
}
```

## JoinStage 5 erstellen: Treten Sie der Stage bei

Vervollständigen wir unsere Funktion `joinStage` , indem Sie endlich die Stage betreten!

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

## LeaveStage erstellen

Definieren Sie die `leaveStage`-Funktion, die die Verlassen-Schaltfläche aufrufen wird.

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

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

## Input-Event-Handler initialisieren

Wir fügen eine letzte Funktion zu unserer `app.js`-Datei hinzu. Diese Funktion wird sofort aufgerufen, wenn die Seite geladen wird, und richtet Event-Handler für den Beitritt und das Verlassen der Stage ein.

```
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
```

## Führen Sie die Anwendung aus und geben Sie ein Token an

Jetzt können Sie die Webseite lokal oder mit anderen teilen, die Seite öffnen, ein Teilnehmer-Token eingeben und der Stufe beitreten.

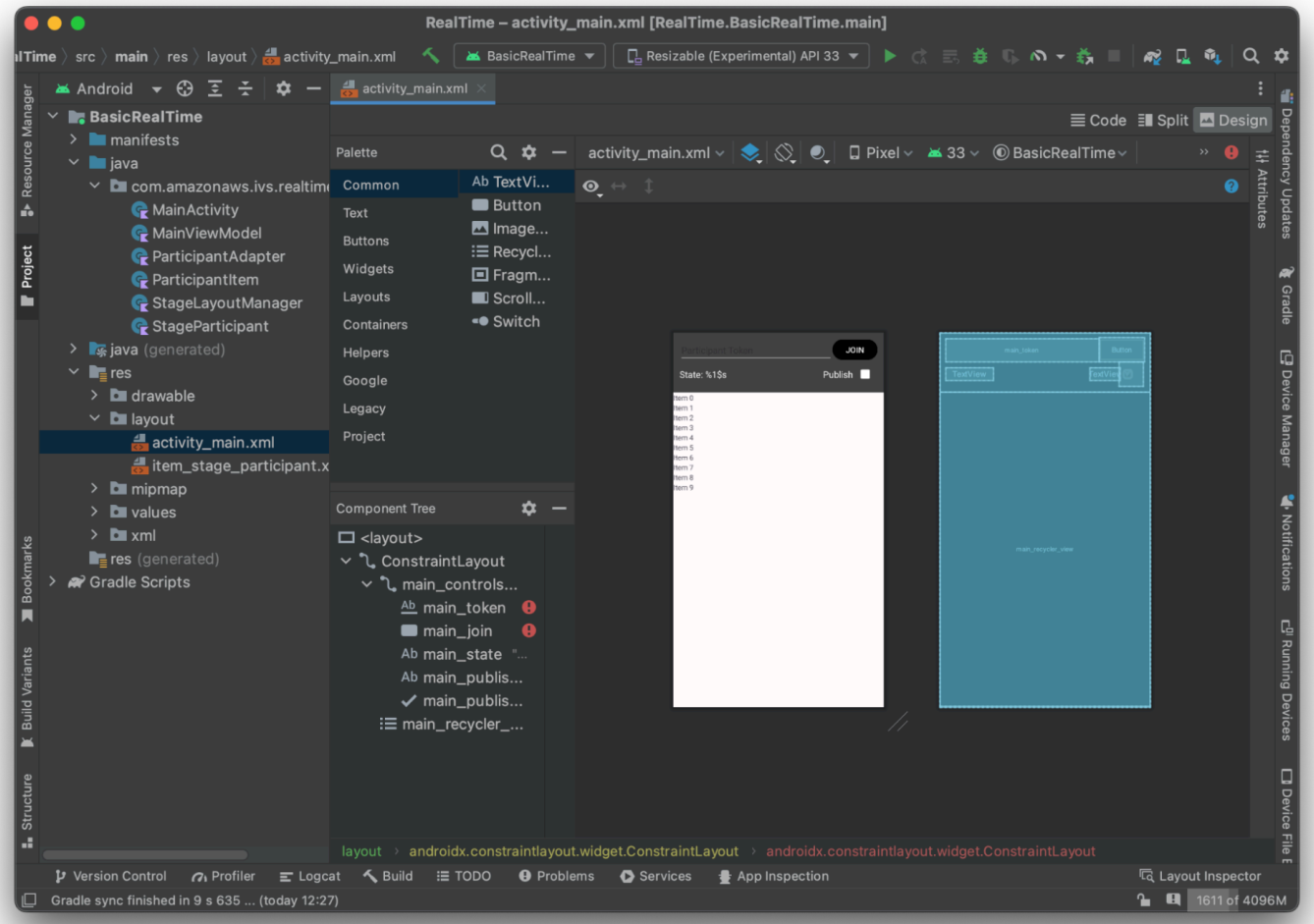
## Die nächsten Themen

Ausführlichere Beispiele für npm, React und mehr finden Sie in IVS-Broadcast-SDK: Web-Leitfaden (Anleitung zu Echtzeit-Streaming).

# Android

## Ansichten erstellen

Wir beginnen mit der Erstellung eines einfachen Layouts für unsere Anwendung mithilfe der automatisch erstellten `activity_main.xml`-Datei. Das Layout enthält einen `EditText`, um ein Token hinzuzufügen, einen Beitreten-`Button`, eine `TextView`, um den Status der Stage anzuzeigen, und eine `CheckBox`, um die Veröffentlichung umzuschalten.




Hier ist das XML hinter der Ansicht:

```
<?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>
```

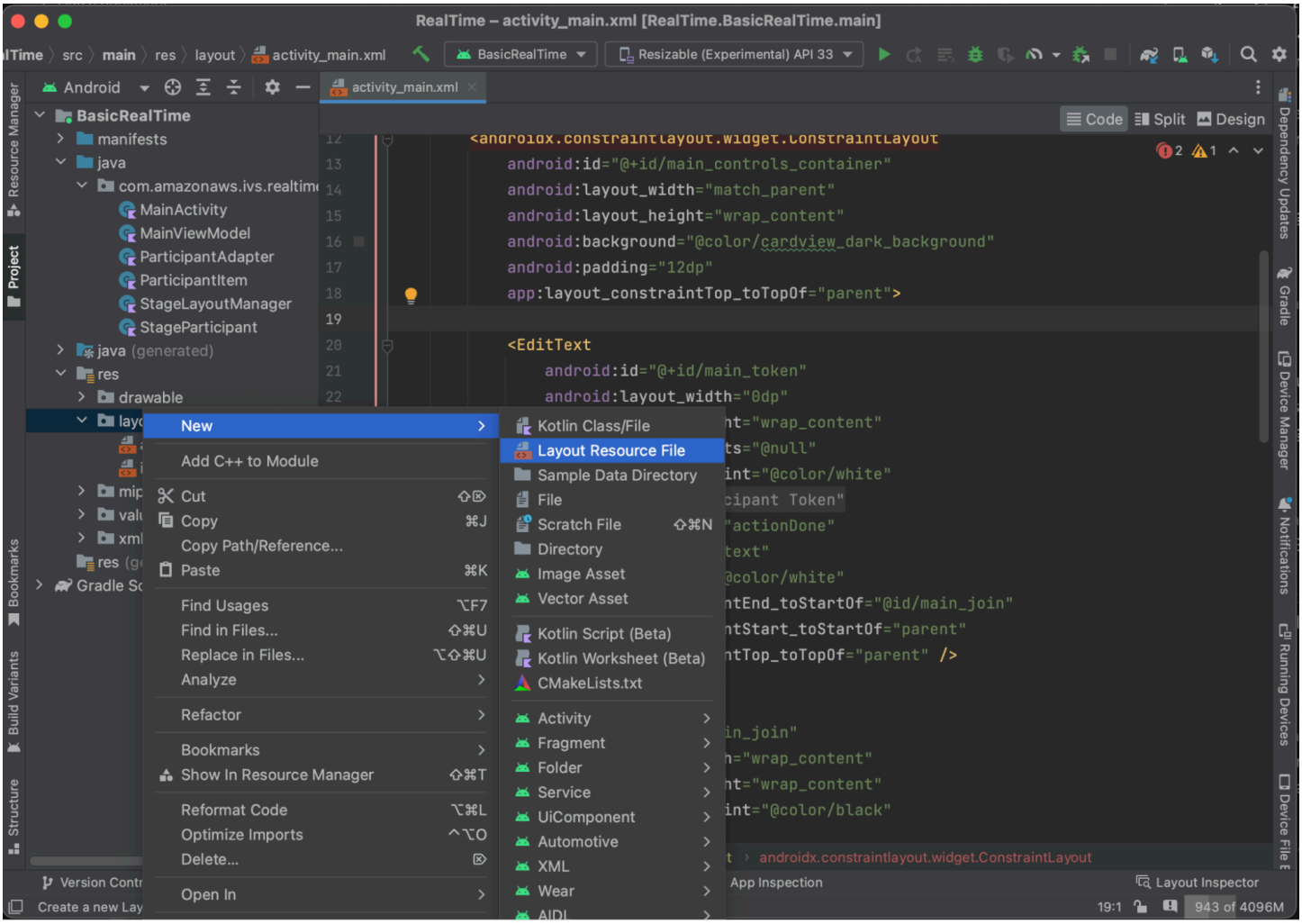
Wir haben hier auf ein paar String-IDs verwiesen, also erstellen wir unsere gesamte `strings.xml`-Datei jetzt:

```
<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>
```

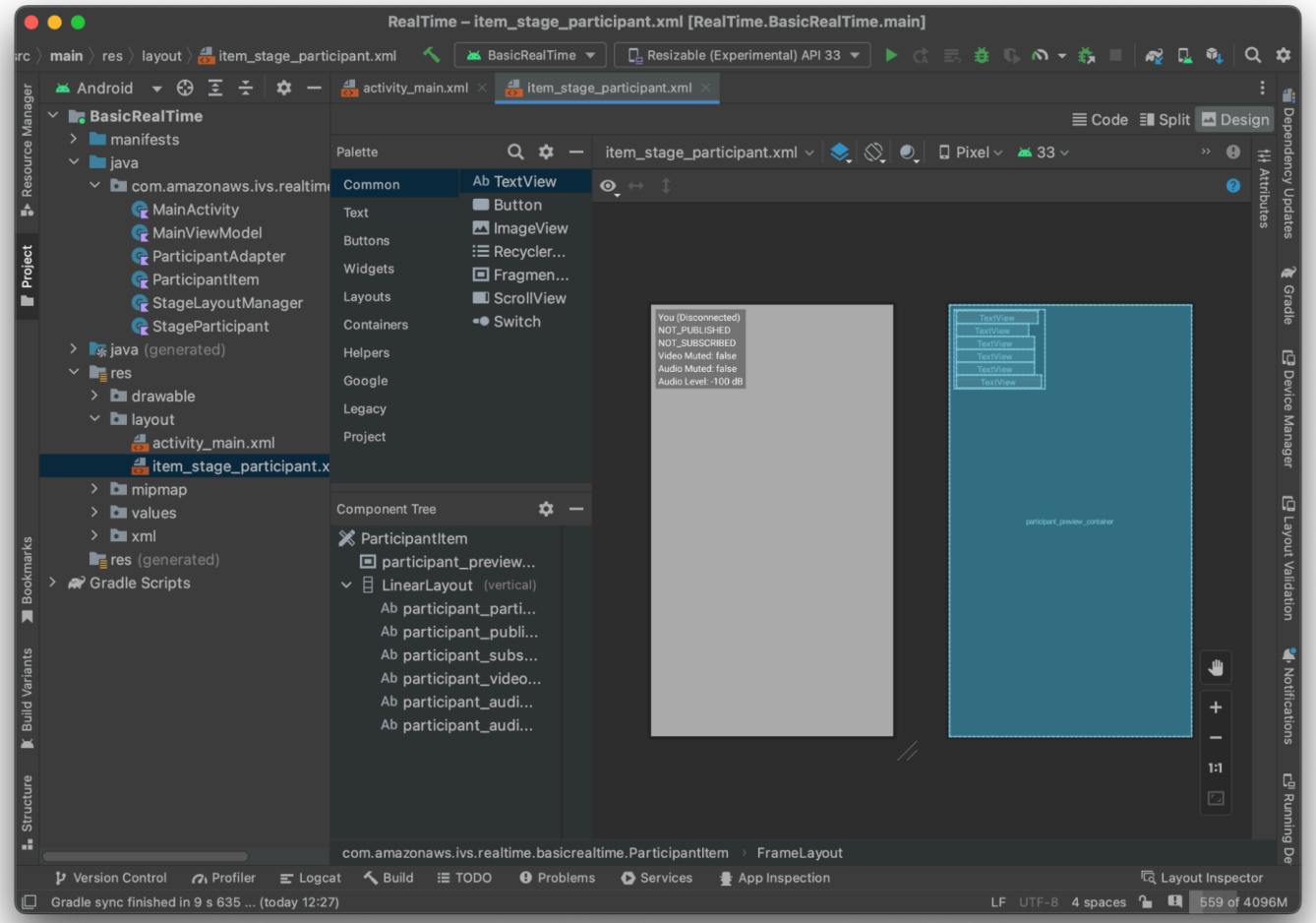
Lassen Sie uns diese Ansichten im XML mit unseren `MainActivity.kt` verknüpfen:

```
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)
}
```

Jetzt erstellen wir eine Elementansicht für unsere `RecyclerView`. Klicken Sie dazu mit der rechten Maustaste auf `res/layout`-Verzeichnis und wählen Sie Neu > Layout-Ressourcendatei. Benennen Sie diese Datei `item_stage_participant.xml`.




Das Layout für dieses Element ist einfach: Es enthält eine Ansicht zum Rendern des Videostreams eines Teilnehmers und eine Liste von Labels zur Anzeige von Informationen über den Teilnehmer:




Hier ist das 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>
```

Diese XML-Datei generiert eine Klasse, die wir noch nicht erstellt haben, `ParticipantItem`. Da das XML den vollständigen Namespace enthält, sollten Sie diese XML-Datei unbedingt in Ihren Namespace aktualisieren. Lassen Sie uns diese Klasse erstellen und die Ansichten einrichten, aber ansonsten lassen wir das Feld vorerst leer.

Erstellen Sie eine neue Kotlin-Klasse, `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)
    }
}
```

## Berechtigungen

Um die Kamera und das Mikrofon verwenden zu können, müssen Sie vom Benutzer Berechtigungen anfordern. Dafür folgen wir einem standardmäßigen Berechtigungsablauf:

```
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
    }
}
```

## Anwendungsstatus

Unsere Anwendung verfolgt die Teilnehmer lokal in einem `MainViewModel.kt` und der Status wird `MainActivity` dem mithilfe von Kotlin zurückgemeldetStateFlow.

Erstellen Sie eine neue Kotlin-Klasse `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` verwalten wir unser Ansicht-Modell:

```
import androidx.activity.viewModels

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

Um `AndroidViewModel` und diese Kotlin-`ViewModel`-Erweiterungen zu verwenden, müssen Sie Folgendes zur `build.gradle`-Datei Ihres Moduls hinzufügen:

```
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 Adapter

Wir werden eine einfache `RecyclerView.Adapter`-Unterklasse erstellen, um unsere Teilnehmer zu verfolgen und unsere `RecyclerView` auf Stageereignisse zu aktualisieren. Aber zuerst brauchen wir eine Klasse, die einen Teilnehmer repräsentiert. Erstellen Sie eine neue Kotlin-Klasse `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)
            }
        }
}
```

Wir verwenden diese Klasse in der `ParticipantAdapter`-Klasse, die wir als Nächstes erstellen werden. Wir beginnen damit, die Klasse zu definieren und eine Variable zu erstellen, um die Teilnehmer zu verfolgen:

```
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>()
```

Wir müssen auch unsere `RecyclerView.ViewHolder` definieren bevor Sie die restlichen Überschreibungen implementieren:

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

Damit können wir die Standard-`RecyclerView.Adapter`-Überschreibung implementieren:

```
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)
    }
}
```

Schließlich fügen wir neue Methoden hinzu, die wir von unserem `MainViewModel` abrufen, wenn Änderungen an den Teilnehmern vorgenommen werden. Bei diesen Methoden handelt es sich um Standard-CRUD-Operationen auf dem Adapter.

```
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])
    }
}
```

Wieder in `MainViewModel` müssen wir einen Verweis auf diesen Adapter erstellen und speichern:

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

## Stufenstatus

Wir müssen auch einige Stagestatus innerhalb `MainViewModel` verfolgen. Definieren wir jetzt diese Eigenschaften:

```
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>()
```

Um Ihre eigene Vorschau zu sehen, bevor Sie eine Stage betreten, erstellen wir sofort einen lokalen Teilnehmer:

```
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)
}
```

Wir wollen sicherstellen, dass wir diese Ressourcen bereinigen, wenn unsere `ViewModel` bereinigt ist. Wir überschreiben on`Cleared()` sofort, damit wir nicht vergessen, diese Ressourcen zu reinigen.

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

Jetzt füllen wir unsere lokale `streams`-Eigenschaft auf, sobald die Berechtigungen erteilt sind, und implementieren die `permissionsGranted`-Methode, die wir zuvor aufgerufen haben:

```
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)
    }
}
```

## Implementierung des Stage-SDK

Drei Kernkonzepte liegen der Echtzeit-Funktionalität zugrunde: Stage, Strategie und Renderer. Das Designziel besteht in der Minimierung der Menge an clientseitiger Logik, die für die Entwicklung eines funktionierenden Produkts erforderlich ist.

### Stage.Strategy

Die `Stage.Strategy`-Implementierung ist einfach:

```
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
}
```

Zusammenfassend lässt sich sagen, dass wir auf der Grundlage unseres internen Status `publishEnabled` veröffentlichen, und wenn wir veröffentlichen, veröffentlichen wir die zuvor gesammelten Streams. Schließlich abonnieren wir für dieses Beispiel immer andere Teilnehmer und erhalten sowohl ihr Audio als auch ihr Video.

StageRenderer

Die `StageRenderer`-Implementierung ist ebenfalls ziemlich einfach, obwohl sie angesichts der Anzahl der Funktionen reichlich mehr Code enthält. Der allgemeine Ansatz in diesem Renderer besteht darin, unseren `ParticipantAdapter` zu aktualisieren, wenn das SDK uns über eine Änderung an einem Teilnehmer informiert. Es gibt bestimmte Szenarien, in denen wir mit lokalen Teilnehmern anders umgehen, weil wir beschlossen haben, sie selbst zu verwalten, sodass sie ihre Kameravorschau sehen können, bevor sie beitreten.

```
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) {}
}
```

## Implementieren eines benutzerdefinierten RecyclerView LayoutManager

Die Festlegung verschiedener Teilnehmerzahlen kann komplex sein. Sie möchten, dass sie den gesamten Frame der übergeordneten Ansicht einnehmen, aber Sie möchten nicht jede Teilnehmerkonfiguration unabhängig voneinander handhaben. Um dies zu vereinfachen, führen wir die Implementierung eines `RecyclerView.LayoutManager` durch.

Erstelle eine weitere neue Klasse `StageLayoutManager`, welche `GridLayoutManager` erweitern soll. In diesem Kurs wird das Layout für jeden Teilnehmer anhand der Anzahl der Teilnehmer in einem flussbasierten Zeilen-/Spaltenlayout berechnet. Jede Zeile hat dieselbe Höhe wie die anderen, aber Spalten können pro Zeile unterschiedlich breit sein. Sehen Sie den Code-Kommentar über der `layouts`-Variable für eine Beschreibung, wie dieses Verhalten angepasst werden kann.

```
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
}
```

Wieder zurück in `MainActivity.kt` müssen wir den Adapter und den Layoutmanager für unsere `RecyclerView` einrichten:

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

## UI-Aktionen verbinden

Wir sind nah dran; es gibt nur ein paar UI-Aktionen, die wir verbinden müssen.

Zuerst haben wir unsere `MainActivity`, die die `StateFlow`-Änderungen von`MainViewModel` beobachtet:

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

Als Nächstes fügen wir Listener zu unseren Schaltflächen „Beitreten“ und „Veröffentlichen“ hinzu:

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

Beide der oben genannten Anruffunktionen in unserer `MainViewModel`, die wir jetzt implementieren:

```
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
}
```

## Rendern der Teilnehmer

Schließlich müssen wir die Daten, die wir vom SDK erhalten, auf das zuvor erstellte Teilnehmerelement übertragen. Wir haben die Logik von `RecyclerView` bereits fertig, also müssen wir nur noch die API von `bind` in `ParticipantItem` implementieren.

Wir beginnen mit dem Hinzufügen der leeren Funktion und gehen sie dann Schritt für Schritt durch:

```
fun bind(participant: StageParticipant) {

}
```

Zuerst kümmern wir uns um den Status „Einfach“, die Teilnehmer-ID, den Veröffentlichungsstatus und den Abonnementstatus. Für diese aktualisieren wir einfach unsere `TextViews` direkt:

```
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
```

Als Nächstes aktualisieren wir die stummgeschalteten Audio- und Videozustände. Um den Stummschaltzustand zu erhalten, müssen wir den `ImageDevice` und `AudioDevice` aus dem Streams-Array finden. Um die Leistung zu optimieren, erinnern wir uns an die zuletzt angehängten Geräte-IDs.

```
// 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"
}
```

Abschließend wollen wir eine Vorschau für das `imageDevice` rendern:

```
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
```

Und wir zeigen Audiostatistiken von der `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
```

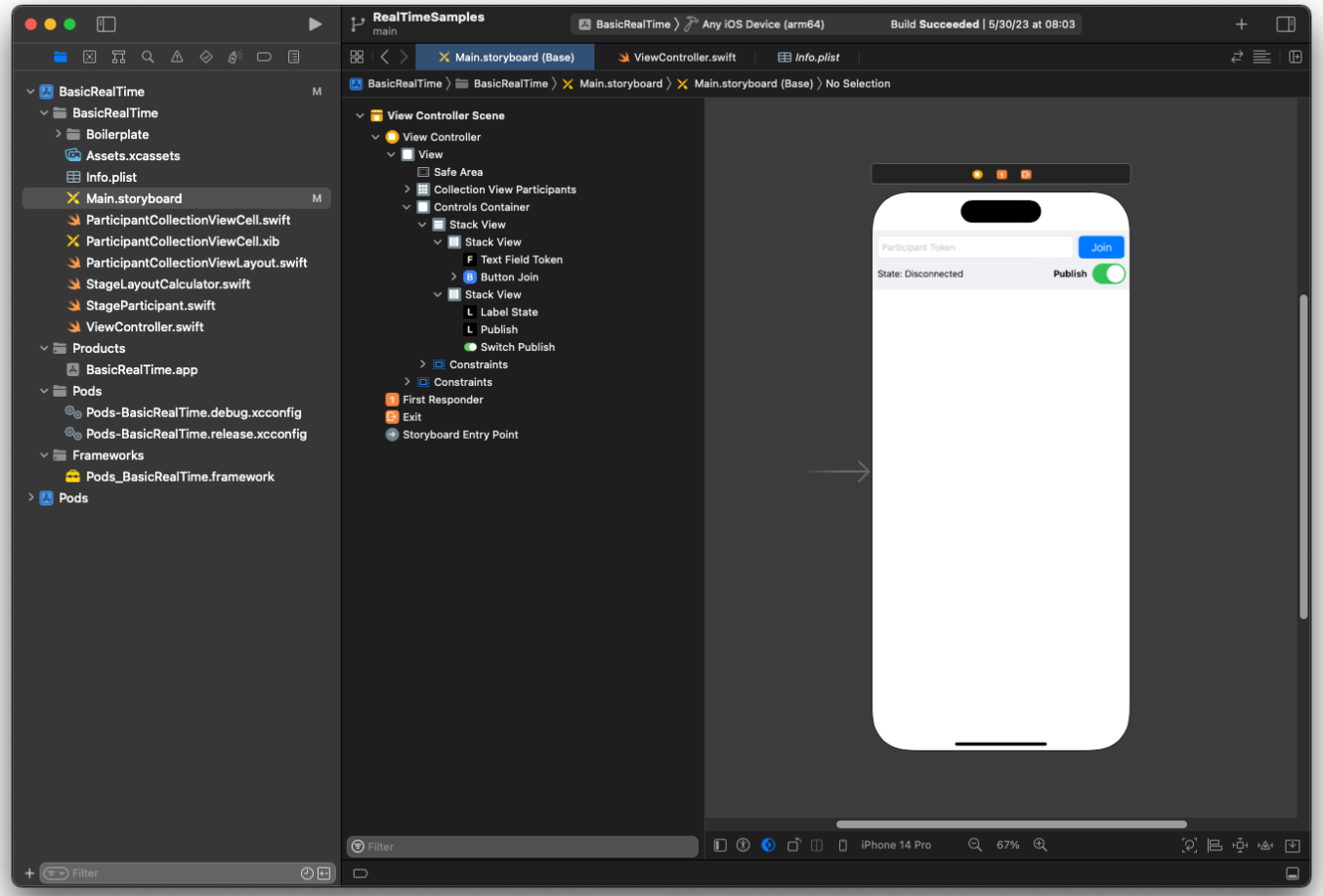
# iOS

## Ansichten erstellen

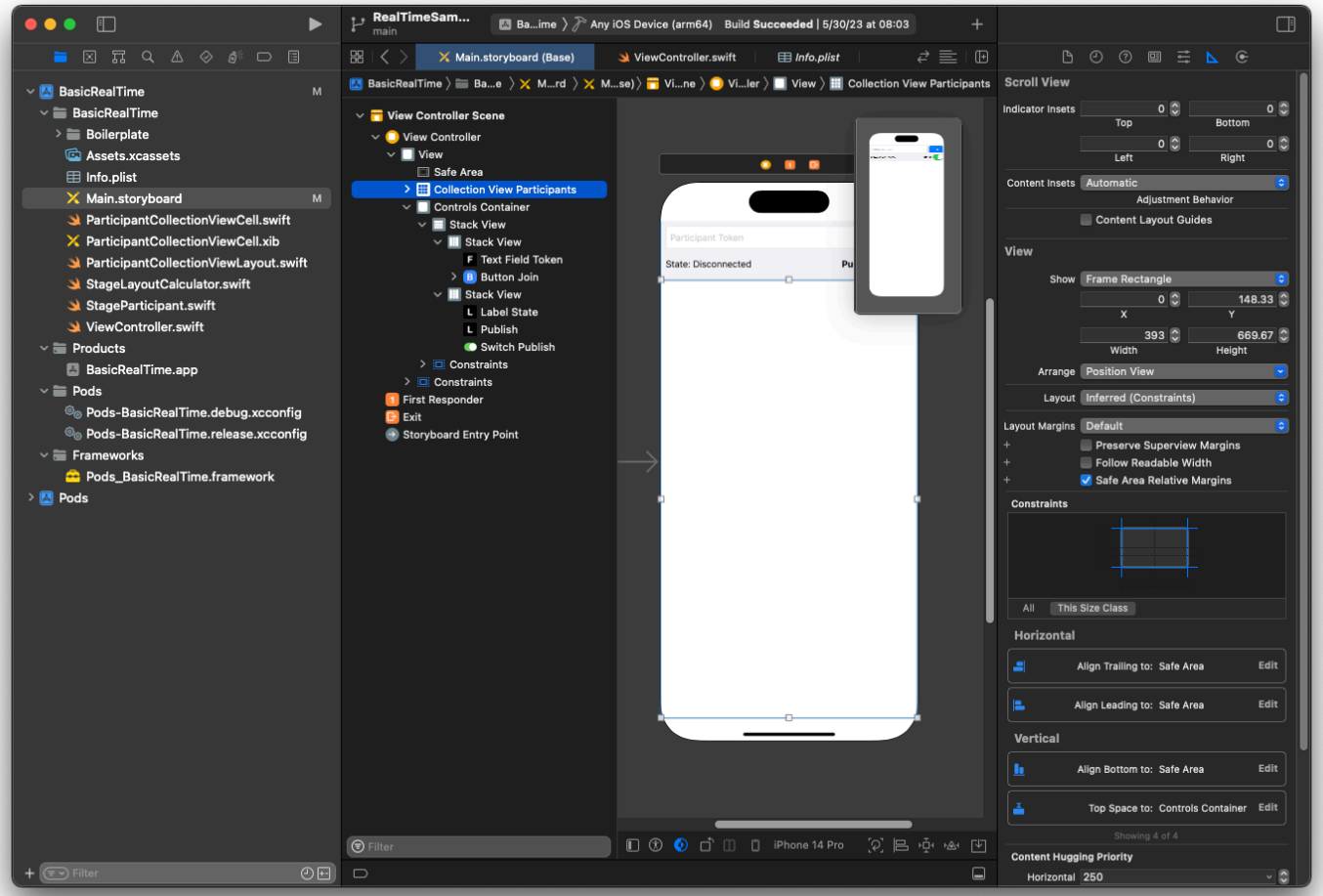
Wir beginnen mit der automatisch erstellten `ViewController.swift`-Datei,
um `AmazonIVSBroadcast` zu importieren und dann fügen wir etwas `@IBOutlets` hinzu zum Verlinken:

```
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!
```

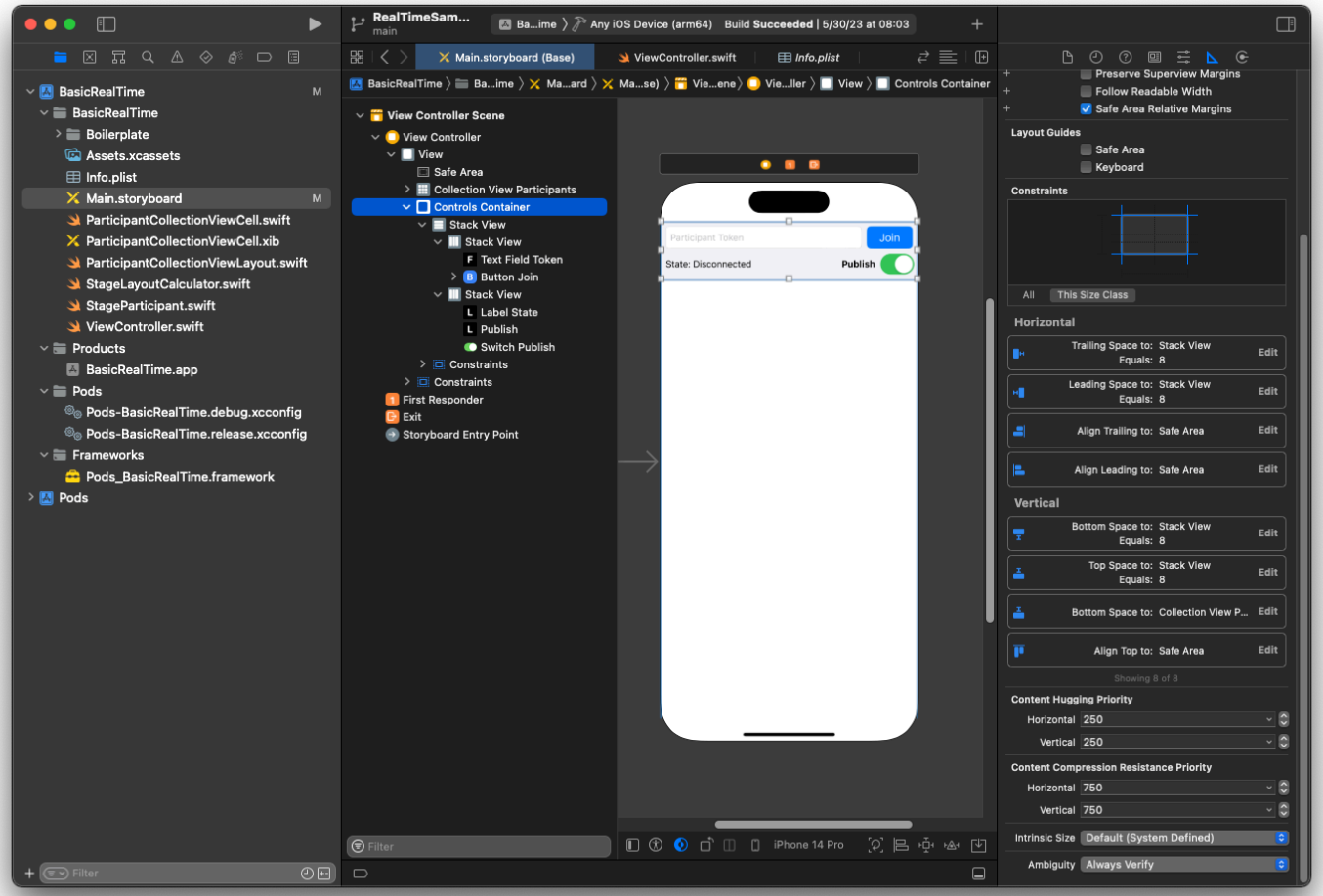
Jetzt erstellen wir diese Ansichten und verknüpfen sie in `Main.storyboard`. Hier ist die Ansichtsstruktur, die wir verwenden werden:



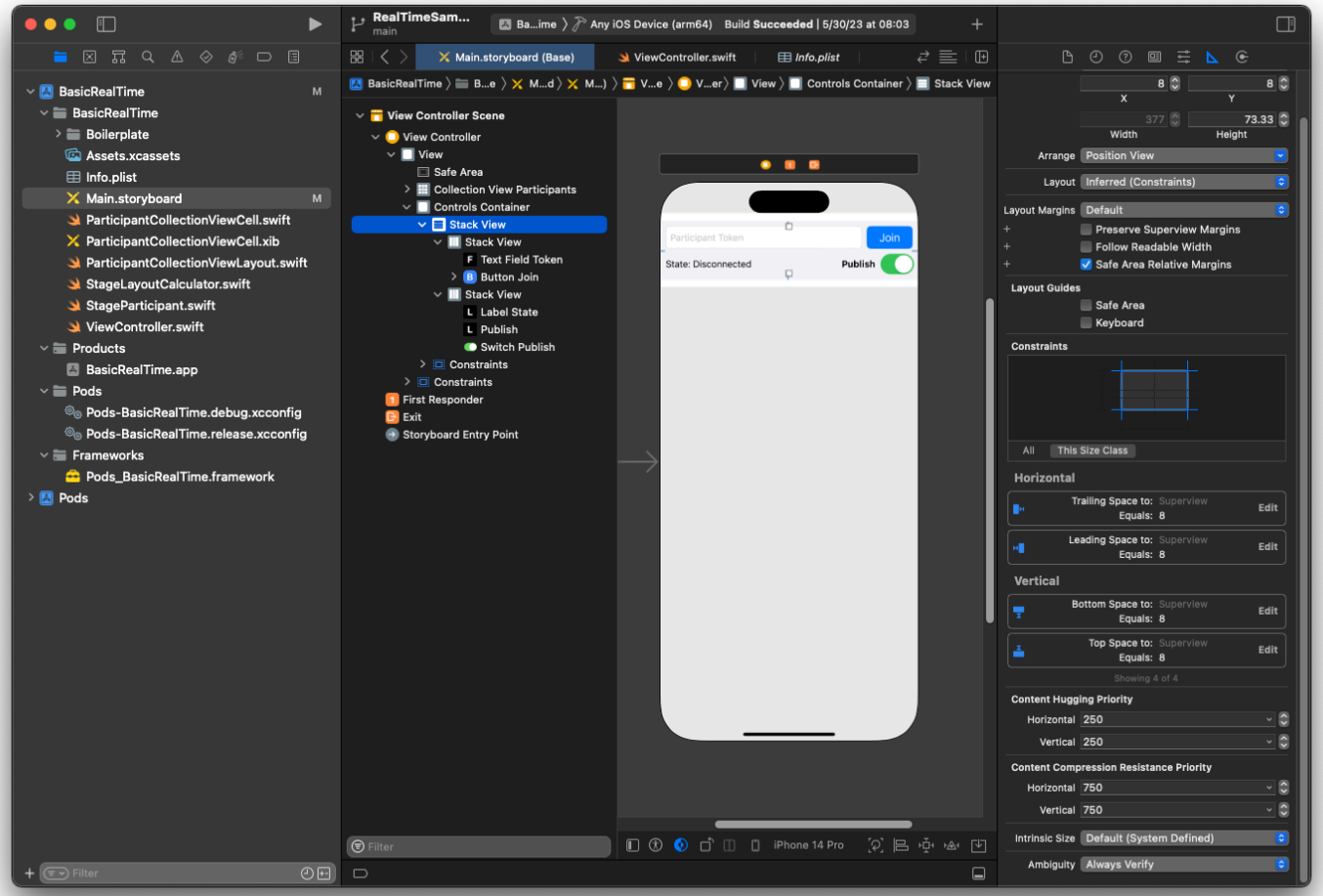

Für die AutoLayout Konfiguration müssen wir drei Ansichten anpassen. Die erste Ansicht ist Sammlungsansicht der Teilnehmer (ein `UICollectionView`). Binden Sie Führend, Verfolgend, und Unterseite zu Sicherer Bereich. Binden Sie auch Oben zu Steuert Container.



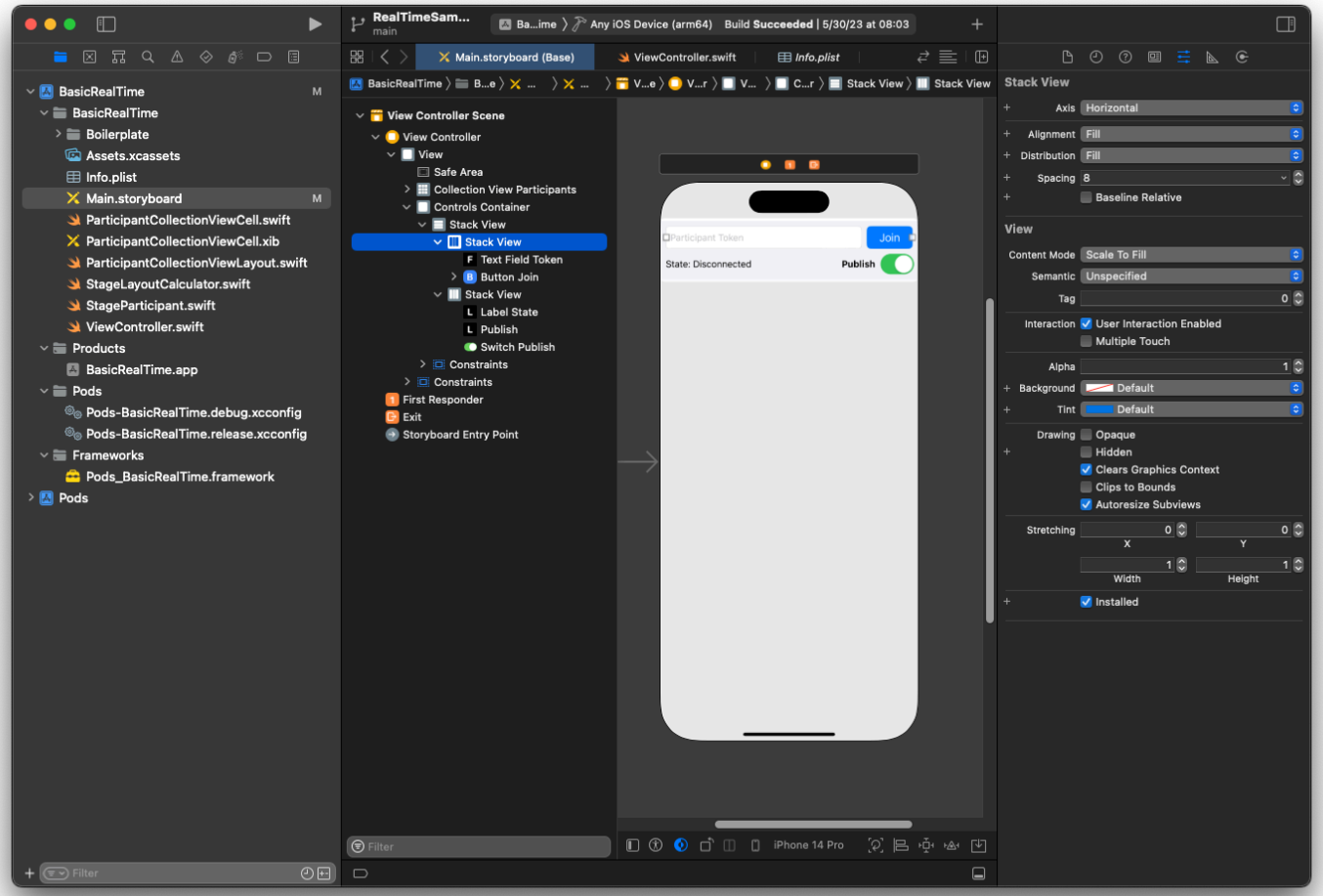

Die zweite Ansicht ist Steuert Container. Binden Sie Führend, Verfolgend, und Unterseite zu Sicherer Bereich:




Die dritte und letzte Ansicht ist Vertikale Stapelansicht. Binden Sie Oben,Führend,Verfolgend, und Unterseite zu Superansicht. Stellen Sie für das Styling den Abstand auf 8 statt auf 0 ein.



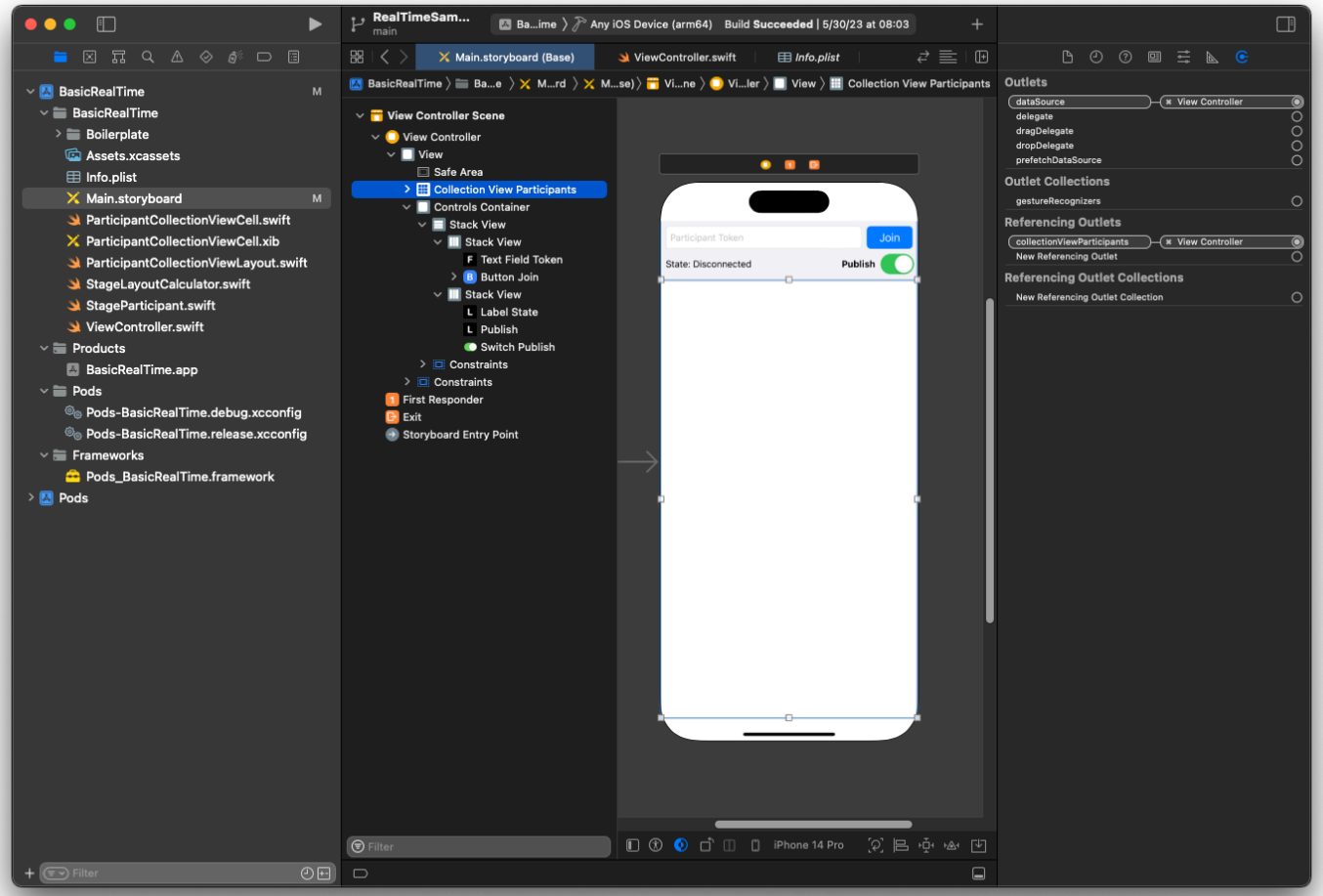

Die BenutzeroberflächeStackViews übernimmt das Layout der verbleibenden Ansichten. Verwenden Sie für alle drei BenutzeroberflächenStackViews Füllen als Ausrichtung und Verteilung .



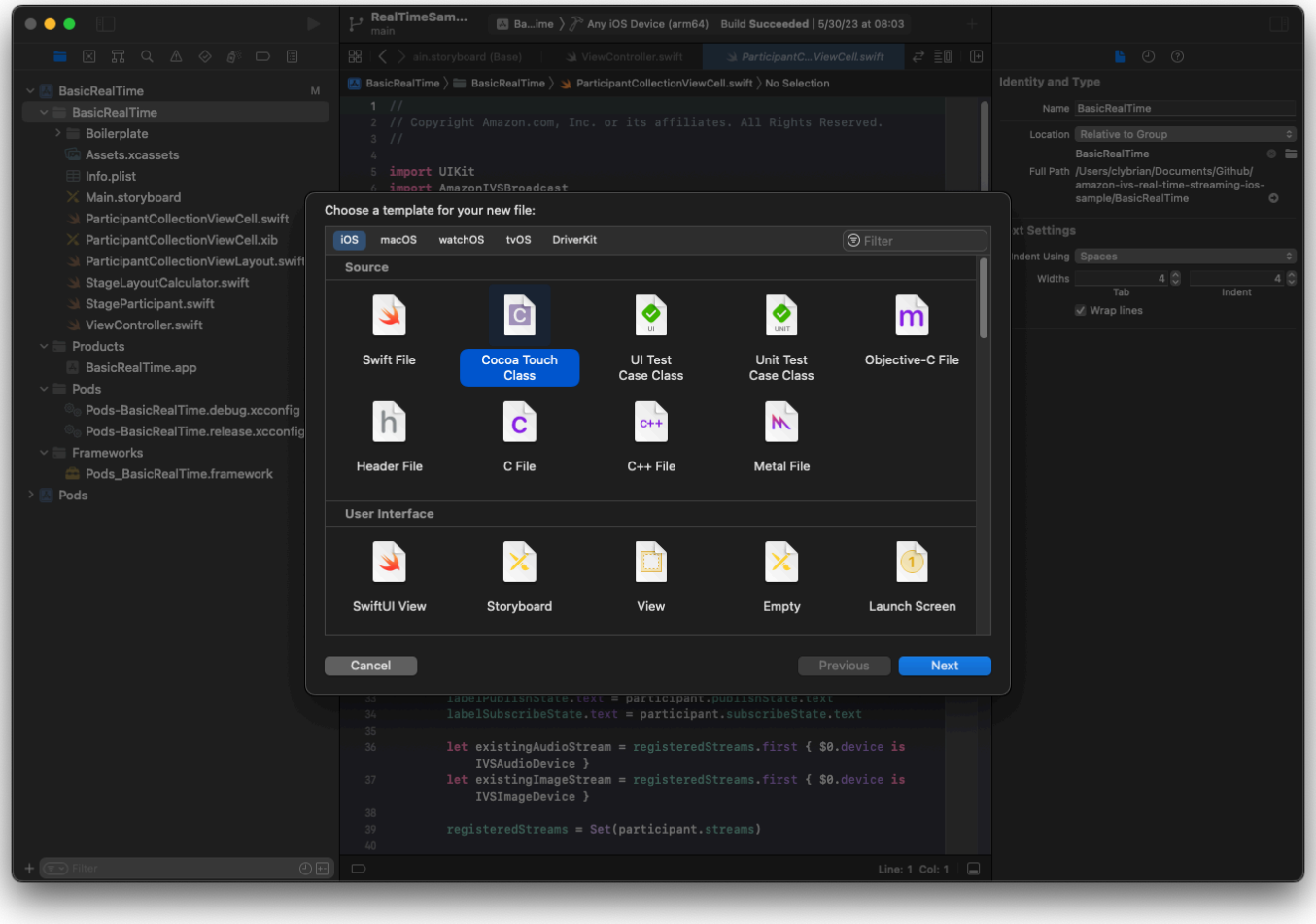

Lassen Sie uns abschließend diese Ansichten mit unserem `ViewController` verknüpfen. Kartieren Sie von oben die folgenden Ansichten:

- Textfeld-Verknüpfung bindet an `textFieldToken`.
- Schaltfläche Beitreten bindet an `buttonJoin`.
- Status beschriften bindet an `labelState`.
- Veröffentlichen wechseln bindet an `switchPublish`.
- Sammlungsansicht der Teilnehmer bindet an `collectionViewParticipants`.

Nutzen Sie diese Zeit auch, um das `dataSource` des Elements Sammlungsansicht der Teilnehmer auf den Besitz von `ViewController` einzustellen:



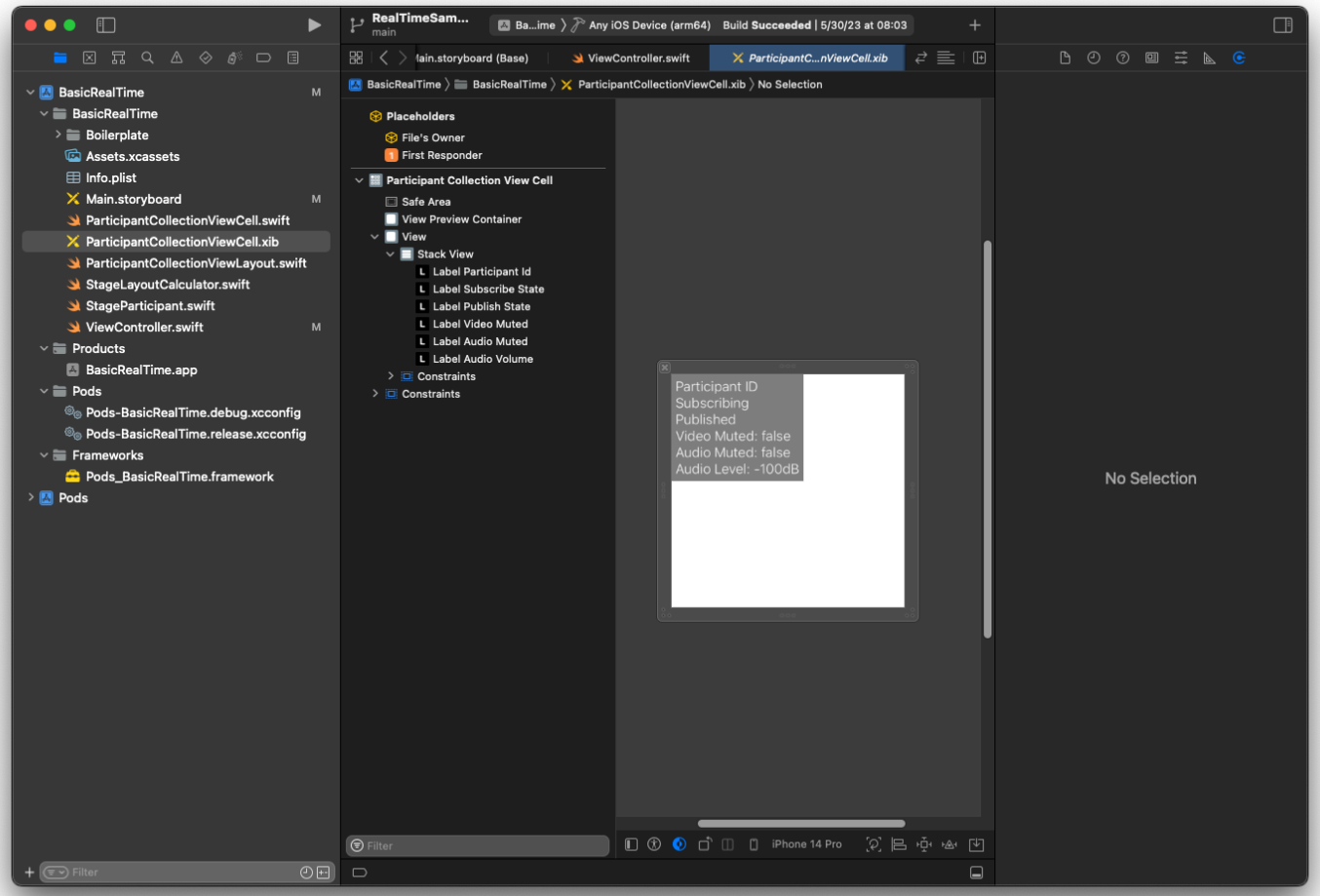

Jetzt erstellen wir die `UICollectionViewCell`-Unterklasse, in welcher die Teilnehmer gerendert werden sollen. Erstellen Sie zunächst eine neue Cocoa-Touch-Class-Datei:



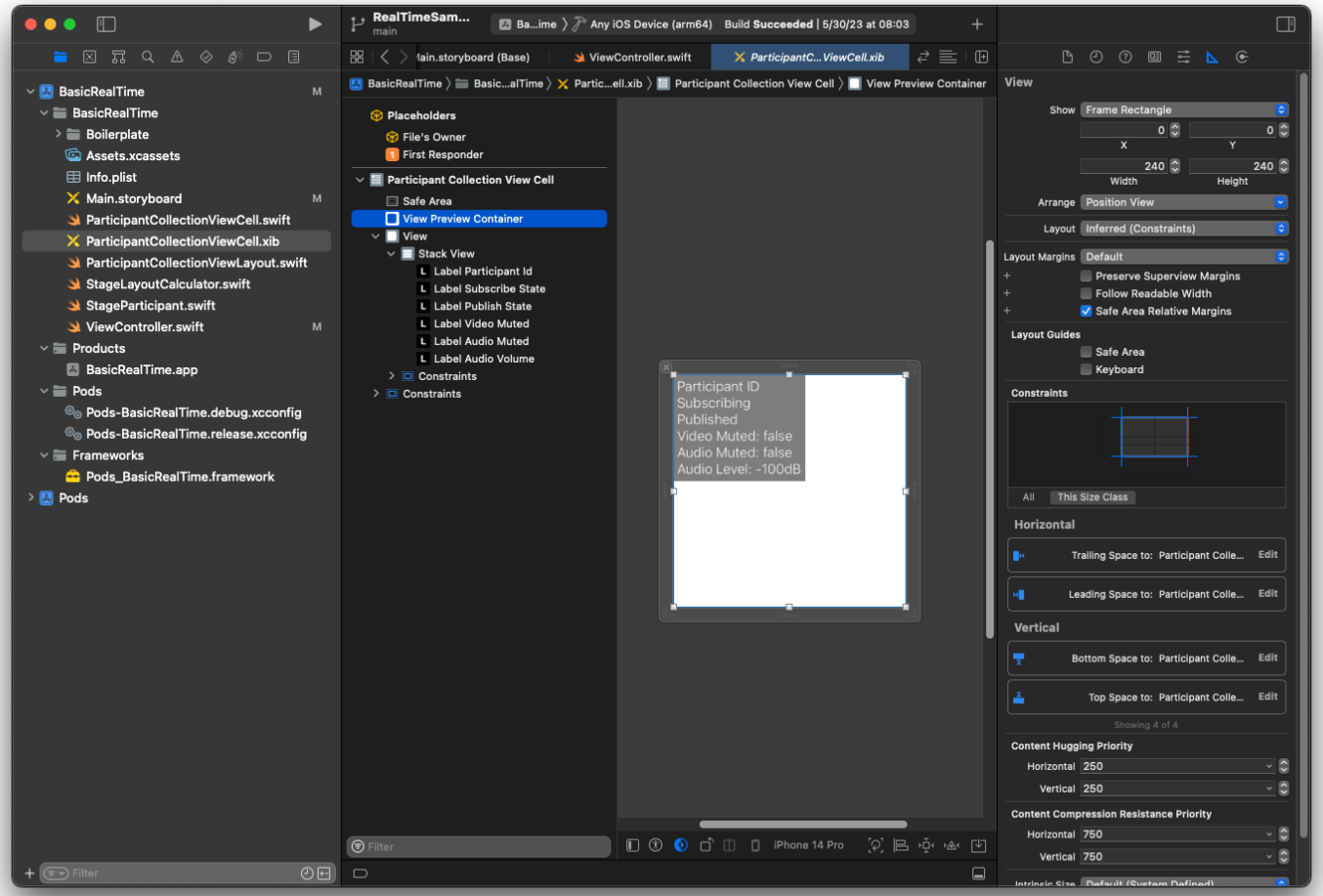

Nennen Sie sie `ParticipantUICollectionViewCell` und machen Sie es zu einer Unterklasse von `UICollectionViewCell` in Swift. Wir beginnen erneut in der Swift-Datei und erstellen unsere `@IBOutlets` zum Verlinken:

```
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!
```

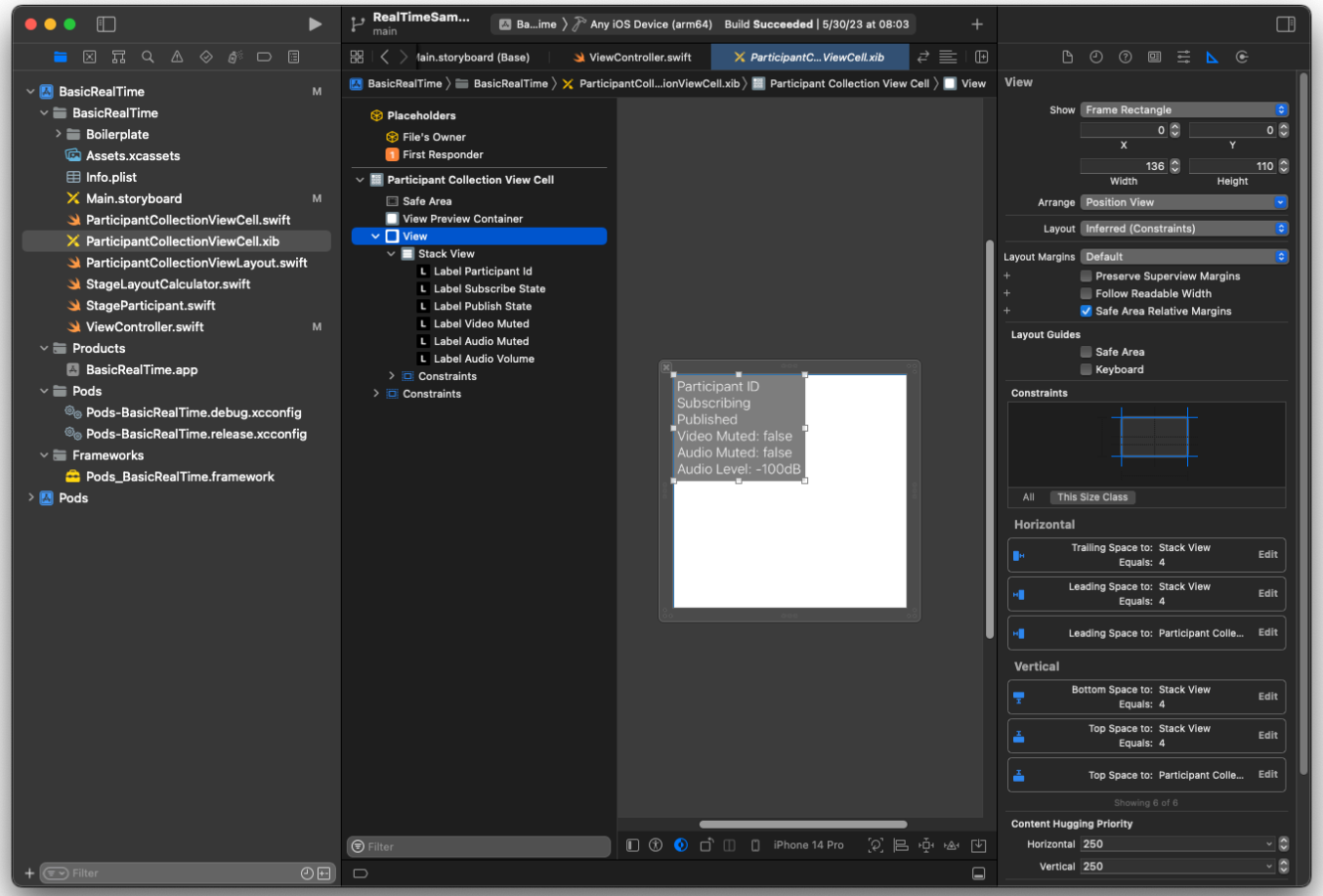
Erstellen Sie in der zugehörigen XIB-Datei diese Ansichtshierarchie:



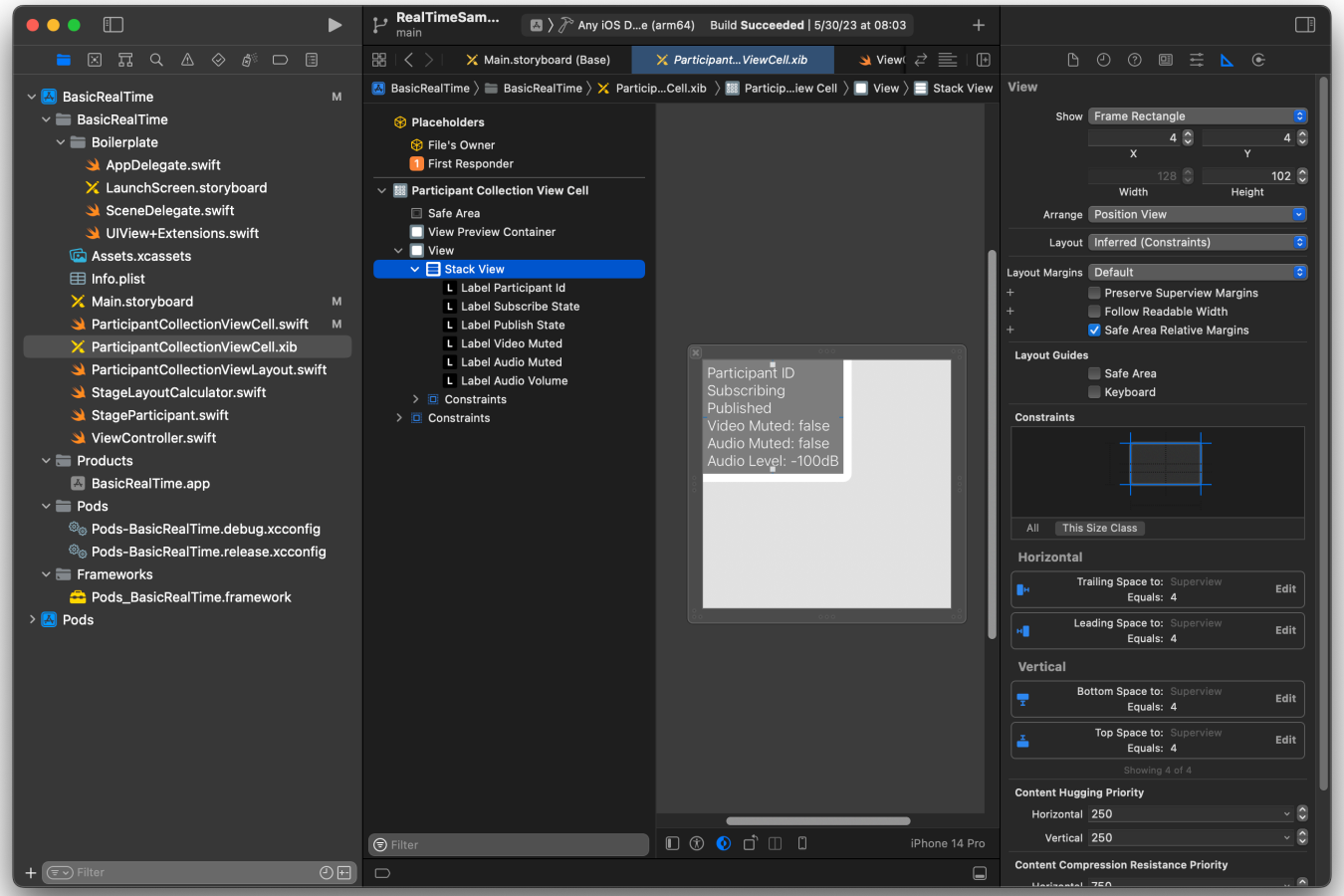

Für ändern AutoLayoutwir erneut drei Ansichten. Die erste Ansicht ist Vorschaucontainer anzeigen. Setzen Sie Verfolgend, Führend, Oben und Unterseite zu Sammlungsansicht der Teilnehmer Zelle.




Die zweite Ansicht ist Ansicht. Setzen Sie Führend und Oben  zu Sammlungsansicht der Teilnehmer Zelle und ändern Sie den Wert auf 4.




Die dritte Ansicht ist Stapelansicht. Setzen
Sie Verfolgend, Führend, Oben und Unterseite zuSuperansicht und ändern Sie den Wert auf 4.




## Berechtigungen und Idle Timer

Zurück zu unserem `ViewController`. Wir werden den System-Leerlauf-Timer deaktivieren, um zu verhindern, dass das Gerät in den Ruhemodus wechselt, während unsere Anwendung verwendet wird:

```
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
}
```

Als Nächstes fordern wir Kamera- und Mikrofonberechtigungen vom System an:

```
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)
    }
}
```

## Anwendungsstatus

Wir müssen unsere `collectionViewParticipants` konfigurieren mit der Layout-Datei, die wir zuvor erstellt haben:

```
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")
}
```

Um jeden Teilnehmer zu repräsentieren, erstellen wir eine einfache Struktur namens `StageParticipant`. Diese kann enthalten sein in der `ViewController.swift`-Datei, oder es kann eine neue Datei erstellt werden.

```
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
    }
}
```

Um diese Teilnehmer zu verfolgen, verwahren wir eine Reihe von ihnen als Privateigentum in unserem `ViewController`:

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

Diese Eigenschaft wird verwendet, um unsere `UICollectionViewDataSource` anzutreiben, die früher  vom Storyboard aus verlinkt wurde:

```
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'")
        }
    }

}
```

Um Ihre eigene Vorschau zu sehen, bevor Sie eine Stage betreten, erstellen wir sofort einen lokalen Teilnehmer:

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

Dies führt dazu, dass sofort nach dem Ausführen der App eine Teilnehmerzelle gerendert wird, die den lokalen Teilnehmer darstellt.

Die Benutzer möchten sich selbst sehen können, bevor sie einer Phase beitreten. Deshalb implementieren wir als Nächstes die `setupLocalUser()`-Methode, die zuvor aus dem Code zur Bearbeitung von Berechtigungen aufgerufen wurde. Wir speichern die Kamera- und Mikrofonreferenz als `IVSLocalStageStream`-Objekte.

```
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)
}
```

Hier haben wir die Kamera und das Mikrofon des Geräts über das SDK gefunden und sie in unserem lokalen `streams`-Objekt gespeichert, dann zum `streams`-Array des ersten Teilnehmers (des lokalen Teilnehmers, den wir zuvor erstellt haben) zu unserem `streams` zugewiesen. Schließlich rufen wir `participantsChanged` mit einem `index` von 0 und `changeType` von `updated` auf. Diese Funktion ist eine Hilfsfunktion für die Aktualisierung unserer `UICollectionView` mit hübschen Animationen. So sieht es aus:

```
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)])
    }
}
```

Machen Sie sich jetzt keine Sorgen über `cell.set`. Darauf kommen wir später zurück, aber dort werden wir den Inhalt der Zelle basierend auf dem Teilnehmer rendern.

Der `ChangeType` ist eine einfache Aufzählung:

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

Schließlich möchten wir verfolgen, ob die Stage angeschlossen ist. Wir verwenden eine einfache `bool`, um das zu verfolgen, wodurch unsere Benutzeroberfläche automatisch aktualisiert wird, wenn sie selbst aktualisiert wird.

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

## Implementierung des Stage-SDK

Drei Kernkonzepte liegen der Echtzeit-Funktionalität zugrunde: Stage, Strategie und Renderer. Das Designziel besteht in der Minimierung der Menge an clientseitiger Logik, die für die Entwicklung eines funktionierenden Produkts erforderlich ist.

### IVSStageStrategy

Die `IVSStageStrategy`-Implementierung ist einfach:

```
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
    }
}
```

Zusammenfassend lässt sich sagen, dass wir nur veröffentlichen, wenn die Option „Veröffentlichen“ aktiviert ist, und wenn wir veröffentlichen, veröffentlichen wir die Streams, die wir zuvor gesammelt haben. Schließlich abonnieren wir für dieses Beispiel immer andere Teilnehmer und erhalten sowohl ihr Audio als auch ihr Video.

IVSStageRenderer

Die `IVSStageRenderer`-Implementierung ist ebenfalls ziemlich einfach, obwohl sie angesichts der Anzahl der Funktionen reichlich mehr Code enthält. Der allgemeine Ansatz in diesem Renderer besteht darin, unser `participants`-Array zu aktualisieren, wenn das SDK uns über eine Änderung an einen Teilnehmer informiert. Es gibt bestimmte Szenarien, in denen wir mit lokalen Teilnehmern anders umgehen, weil wir beschlossen haben, sie selbst zu verwalten, sodass sie ihre Kameravorschau sehen können, bevor sie beitreten.

```
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)
    }
}
```

Dieser Code verwendet eine Erweiterung, um den Verbindungsstatus in menschenfreundlichen Text umzuwandeln:

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

## Implementieren einer benutzerdefinierten BenutzeroberflächeCollectionViewLayout

Die Festlegung verschiedener Teilnehmerzahlen kann komplex sein. Sie möchten, dass sie den gesamten Frame der übergeordneten Ansicht einnehmen, aber Sie möchten nicht jede Teilnehmerkonfiguration unabhängig voneinander handhaben. Um dies zu vereinfachen, führen wir die Implementierung eines `UICollectionViewLayout` durch.

Erstellen Sie eine weitere neue Datei, `ParticipantCollectionViewLayout.swift`, welche `UICollectionViewLayout` erweitern soll. Diese Klasse verwendet eine andere Klasse namens `StageLayoutCalculator`, was wir bald behandeln werden. Die Klasse erhält berechnete Rahmenwerte für jeden Teilnehmer und generiert dann die notwendigen `UICollectionViewLayoutAttributes`-Objekte.

```
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))
            }
        }
    }
```

```
}
```

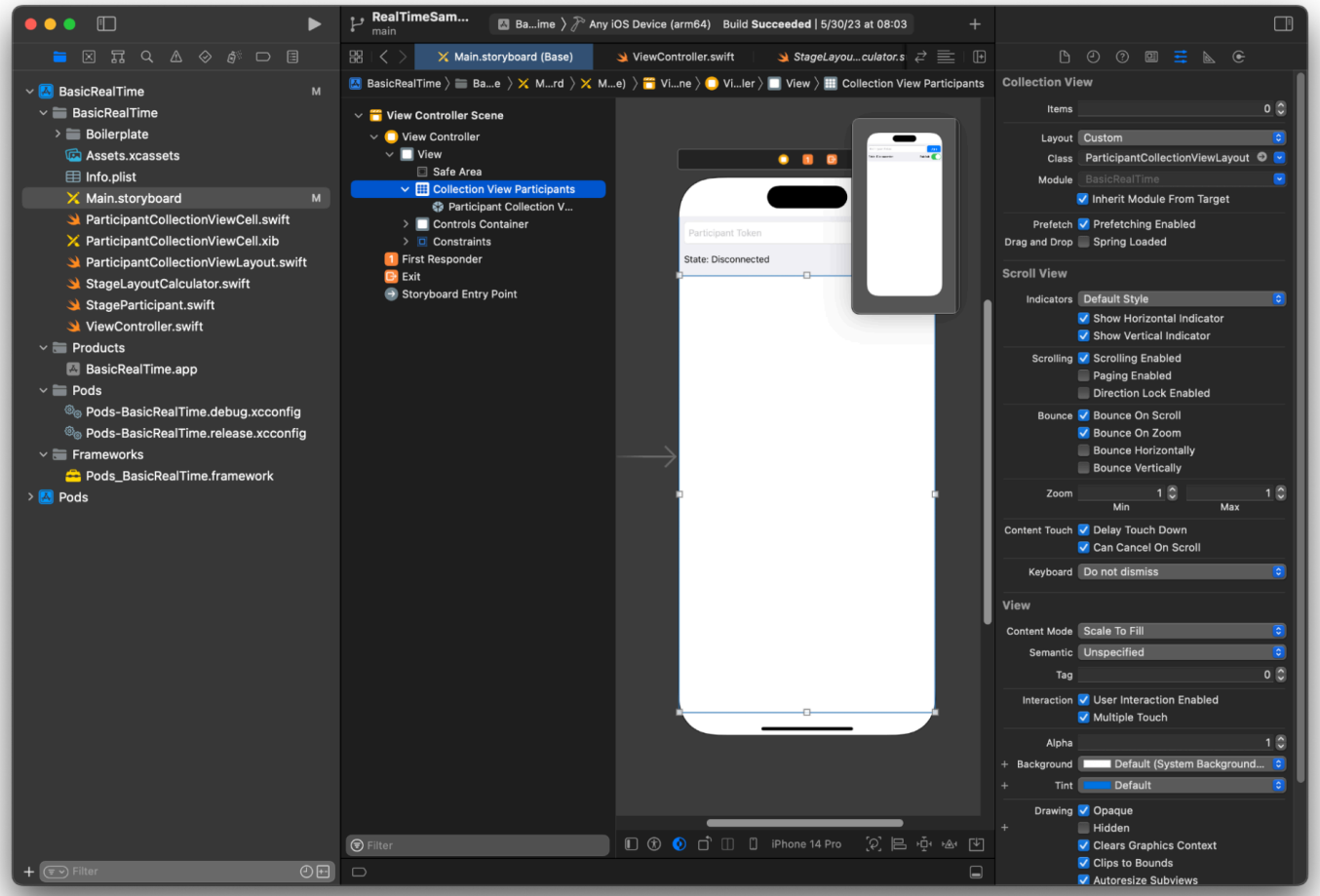
Wichtiger ist die `StageLayoutCalculator.swift`-Klasse. Es ist so konzipiert, dass es die Rahmen für jeden Teilnehmer auf der Grundlage der Anzahl der Teilnehmer in einem fließenden Zeilen-/Spaltenlayout berechnet. Jede Zeile hat dieselbe Höhe wie die anderen, aber Spalten können pro Zeile unterschiedlich breit sein. Sehen Sie den Code-Kommentar über der `layouts`-Variable für eine Beschreibung, wie dieses Verhalten angepasst werden kann.

```
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
    }

}
```

Wieder in `Main.storyboard`, stellen Sie sicher, dass Sie die Layoutklasse für die `UICollectionView` festlegen zu der Klasse, die wir gerade erstellt haben:




## UI-Aktionen verbinden

Wir sind nah dran, es gibt ein paar `IBActions` die wir erstellen müssen.

Zuerst kümmern wir uns um die „Beitreten“-Schaltfläche. Sie reagiert unterschiedlich je nach Wert von `connectingOrConnected`. Wenn sie bereits angeschlossen ist, verlässt sie einfach die Stage. Wenn die Verbindung unterbrochen ist, liest sie den Text aus dem Token `UITextFieldund` schafft eine neue `IVSStage` mit diesem Text. Dann fügen wir unseren `ViewController` hinzu als `strategy`, `errorDelegate`, und Renderer für `IVSStage`, und schließlich treten wir asynchron der Stage bei.

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

Die andere UI-Aktion, die wir anschließen müssen, ist der Publish-Switch:

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

## Rendern der Teilnehmer

Schließlich müssen wir die Daten, die wir vom SDK erhalten, in die Teilnehmerzelle rendern, die wir zuvor erstellt haben. Wir haben die Logik von `UICollectionView` bereits fertig, also müssen wir nur noch die API von `set` in `ParticipantCollectionViewCell.swift` implementieren.

Wir beginnen mit dem Hinzufügen der `empty`-Funktion und gehen Sie sie dann Schritt für Schritt durch:

```
func set(participant: StageParticipant) {

}
```

Zuerst kümmern wir uns um den Status „Einfach“, die Teilnehmer-ID, den Veröffentlichungsstatus und den Abonnementstatus. Für diese aktualisieren wir einfach unsere `UILabels` direkt:

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

Die Texteigenschaften der Publish- und Subscribe-Enums stammen aus lokalen Erweiterungen:

```
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()
        }
    }
}
```

Als Nächstes aktualisieren wir die stummgeschalteten Audio- und Videozustände. Um den Stummschaltzustand zu erhalten, müssen wir den `IVSImageDevice` und `IVSAudioDevice` aus dem `streams`-Array finden. Um die Leistung zu optimieren, erinnern wir uns an die zuletzt angehängten Geräte-IDs.

```
// 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)"
```

Abschließend wollen wir eine Vorschau für das `imageDevice` rendern und Audiostatistiken der `audioDevice` anzeigen:

```
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"
}
```

Die letzte Funktion, die wir erstellen müssen, ist `updatePreview()`, was unserer Ansicht eine Vorschau des Teilnehmers hinzufügt:

```
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)
        }
    }
}
```

Das Obige verwendet eine Hilfsfunktion auf `UIView`, um das Einbetten von Unteransichten zu vereinfachen:

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

# Überwachung von Amazon IVS Echtzeit-Streaming

## Was ist eine Bühnensitzung?

Eine Bühnensitzung beginnt, wenn der erste Teilnehmer eine Bühne betritt und endet einige Minuten, nachdem der letzte Teilnehmer die Veröffentlichung auf der Bühne beendet hat. Bühnensitzungen helfen bei der Fehlerbehebung in langfristigen Bühnen, indem Ereignisse und Teilnehmer in kurzlebige Sitzungen aufgeteilt werden.

## Bühnensitzungen und Teilnehmer anzeigen

### Anleitung für die Konsole

1. Öffnen Sie die Amazon-IVS-Konsole.

   (Sie können auf die Amazon-IVS-Konsole auch über die AWS-Managementkonsole zugreifen.)
2. Klicken Sie im Navigationsbereich auf Bühnen. (Wenn der Navigationsbereich eingeklappt ist, öffnen Sie es zunächst, indem Sie das Hamburger-Symbol auswählen.)
3. Wählen Sie die Bühne aus, um ihre Detailseite aufzurufen.
4. Scrollen Sie auf der Seite nach unten, bis Sie den Abschnitt Bühnensitzungen sehen, und wählen Sie dann eine Bühnensitzung aus, um die zugehörige Detailseite aufzurufen.
5. Um die Teilnehmer der Sitzung anzuzeigen, scrollen Sie nach unten, bis Sie den Abschnitt Teilnehmer sehen. Wählen Sie dann einen Teilnehmer aus, um dessen Detailseite anzuzeigen, einschließlich Diagrammen für Amazon-CloudWatch-Metriken.

## Ereignisse für einen Teilnehmer anzeigen

Ereignisse werden gesendet, wenn sich der Status eines Teilnehmers in einer Stage ändert, z. B. wenn er einer Stage beitritt oder beim Versuch, auf einer Stage zu veröffentlichen, ein Fehler auftritt. Nicht alle Fehler verursachen Ereignisse; z. B. werden clientseitige Netzwerkfehler und Tokensignaturfehler nicht als Ereignisse gesendet. Um diese Fehler in Ihrer Client-Anwendung zu behandeln, verwenden Sie dieIVS-Broadcast-SDKs.

## Anleitung für die Konsole

1. Navigieren Sie wie oben beschrieben zur Seite mit den Teilnehmerdetails.
2. Scrollen Sie nach unten, bis Sie den Abschnitt Ereignisse sehen. Daraufhin wird eine geordnete Liste der Teilnehmer-Ereignisse angezeigt. Unter Verwendung von Amazon EventBridge mit Amazon IVS finden Sie Details zu Ereignissen, die für Teilnehmer ausgegeben werden.

## CLI-Anweisungen

Der Zugriff auf Stagesitzungs-Ereignisse mit der AWS-CLI ist eine erweiterte Option und erfordert, dass Sie zuerst die CLI auf Ihrem Computer herunterladen und konfigurieren. Informationen zu den ersten Schritten finden Sie im AWS-Benutzerhandbuch für die Befehlszeilenschnittstelle.

1. Listen Sie die Bühnensitzungen auf, um eine Bühnensession zu finden:

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

2. Listen Sie die Teilnehmer für eine Bühnensitzung auf, um einen Teilnehmer zu finden:

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

3. Ereignisse für eine Bühnensitzung und einen Teilnehmer auflisten:

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

Hier ist eine Beispielantwort auf den `list-participant-events`-Aufruf:

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

# Zugreifen auf CloudWatch-Metriken

Damit die CloudWatch-Metriken verfügbar sind, sind die folgenden IVS-Broadcast-SDK-Versionen erforderlich: Web 1.5.0 oder höher, Android 1.12.0 oder höher, oder iOS 1.12.0 oder höher.

## Anleitung für die CloudWatch-Konsole

1. Öffnen Sie die CloudWatch-Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Erweitern Sie in der Seitennavigation das Dropdown Metriken und wählen Sie dannAlle Metriken aus.
3. Wählen Sie in der Registerkarte Durchsuchen über das unbeschriftete Dropdown-Menü auf der linken Seite Ihre Heimatregion aus, in der Ihre Kanäle erstellt wurden. Weitere Informationen zu Regionen finden Sie unter Globale Lösung, regionale Kontrolle. Eine Liste der unterstützten Regionen finden Sie auf der Amazon-IVS-Seite in der Allgemeinen AWS-Referenz.
4. Wählen Sie unten in der Registerkarte Durchsuchen den IVSRealTime-Namespace aus.
5. Führen Sie eine der folgenden Aktionen aus:
   a. Geben Sie in der Suchleiste Ihre Ressourcen-ID (Teil der ARN, `arn:::ivs:stage/<resource id>`) ein.

      Wählen Sie dann IVSRealTime > Stufenmetriken aus.
   b. Wenn IVSRealTime als auswählbarer Service unter AWS Namespaces erscheint, wählen Sie ihn aus. Er wird aufgeführt, wenn Sie Amazon-IVS-Streaming in Echtzeit verwenden und es

Metriken an Amazon CloudWatch sendet. (Wenn IVSRealTime nicht aufgeführt ist, haben Sie keine Amazon IVS-Metriken.)

Wählen Sie dann nach Bedarf eine Dimensionsgruppierung aus. Die verfügbaren Dimensionen sind unten in CloudWatch-Metriken aufgeführt.

6. Wählen Sie Metriken aus, die dem Diagramm hinzugefügt werden sollen. Verfügbare Metriken sind unten unter CloudWatch-Metriken aufgeführt.

Sie können auch auf der Detailseite des Stream-Vortrags auf das CloudWatch-Diagramm Ihres Stream-Vortrags zugreifen, indem Sie das Feld Anzeigen in CloudWatch auswählen.

### CLI-Anweisungen

Sie können auf die Metriken auch über die AWS CLI zugreifen. Dies erfordert, dass Sie zuerst die CLI auf Ihrem Computer herunterladen und konfigurieren. Informationen zu den ersten Schritten finden Sie im Benutzerhandbuch für die AWS-Befehlszeilenschnittstelle.

So greifen Sie dann über AWS CLI auf das Amazon-IVS-Streaming in Echtzeit zu:

- Führen Sie an der Eingabeaufforderung Folgendes aus:

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

Weitere Informationen finden Sie unter Amazon CloudWatch verwenden im Amazon CloudWatch-Benutzerhandbuch.

## CloudWatch-Metriken: IVS-Echtzeit-Streaming

Amazon IVS bietet die folgenden Metriken im AWS/IVSRealTime-Namespace.

Damit CloudWatch-Metriken verfügbar sind, muss Web Broadcast SDK 1.5.2 oder höher verwendet werden.

Die Dimension kann die folgenden gültigen Werte haben:

- Die `Stage`-Dimension ist eine Ressourcen-ID (Teil des ARN, `arn:::stage/<resource id>`).
- Die `Participant`-Dimension ist eine `participantID`.

- Das `SimulcastLayer` ist „hi“, „mid“, „low“ oder „no-rid“ für einen `MediaType` von „Video“ oder „disabled“ für einen `MediaType` von „Audio“. Dieser Wert kann auch leer sein.
- Die `MediaType`-Dimension ist „Video“ oder „Audio“ (Zeichenfolge).

| Kennzahl | Dimension | Beschreibung |
|---|---|---|
| `DownloadPacketLoss` | `Stage` | Jedes Beispiel stellt den Prozentsatz der Pakete dar, die von einem bestimmten Subscriber beim Herunterladen vom IVS-Server verloren gegangen sind.<br><br>Einheit: Prozent<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Paketverluste über das konfigurierte Intervall. |
| `DownloadPacketLoss` | `Stage,Participant` | Filtert `DownloadPacketLoss` nach Teilnehmer, für Subscriber, die auch Publisher sind. Die Beispiele stellen den Prozentsatz der Pakete dar, die vom Subscriber während des Herunterladens vom IVS-Server verloren gingen. Beispiele werden nur ausgegeben, wenn der Teilnehmer auch ein Publisher ist.<br><br>Einheit: Prozent<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Frameverluste über das konfigurierte Intervall. |
| `DroppedFrames` | `Stage` | Jedes Sample stellt den Prozentsatz der Frames dar, die von einem bestimmten Subscriber gelöscht wurden.<br><br>Einheit: Prozent<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Frameverluste über das konfigurierte Intervall. |

| Kennzahl | Dimension | Beschreibung |
| --- | --- | --- |
| `DroppedFrames` | `Stage,Participant` | Filtert `DroppedFrames` nach Teilnehmer, für Subscriber, die auch Publisher sind. Die Beispiele stellen den Prozentsatz der Frames dar, die zwischen dem Subscriber und allen Publishern in der Stufe verworfen wurden. Beispiele werden nur ausgegeben, wenn der Teilnehmer auch ein Publisher ist.<br><br>Einheit: Prozent<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Frameverluste über das konfigurierte Intervall. |
| `PublishBitrate` | `Stage` | Die ausgegebenen Samples stellen die Gesamtrate dar, mit der ein bestimmter Publisher sowohl Video- als auch Audiodaten sendet (summiert über alle Simulcast-Ebenen).<br><br>Einheit: Bits pro Sekunde<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Bitrate über das konfigurierte Intervall. |
| `PublishBitrate` | `Stage, Participant, SimulcastLayer, MediaType` | Filtert `PublishBitrate` nach Teilnehmer, Simulcast-Ebene und Medientyp. Die Simulcast-Layer-ID wird vom Broadcast-SDK festgelegt. Wenn Simulcast deaktiviert ist, wird diese Layer-ID auf „deaktiviert“ gesetzt. Der Medientyp ist entweder Video oder Audio.<br><br>Einheit: Bits pro Sekunde<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Bitrate über das konfigurierte Intervall. |

| Kennzahl | Dimension | Beschreibung |
| --- | --- | --- |
| `Publishers` | `Stage` | Anzahl der Teilnehmer, die auf der Stufe veröffentlichen.<br><br>Einheit: Anzahl<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum |
| `PublishResolution` | `Stage, Participant, SimulcastLayer, MediaType` | Anzahl der Pixel in der Breite oder Höhe des Frames, je nachdem, welcher Wert kleiner ist. Für einen Frame im Querformat mit einer Größe von 1920x1080 beträgt die PublishResolution beispielsweise 1080. Für einen Frame im Hochformat mit einer Größe von 720x1280 beträgt die PublishResolution 720.<br><br>Einheit: Anzahl<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum |
| `SubscribeBitrate` | `Stage` | Die ausgegebenen Samples stellen die Gesamtrate dar, mit der ein bestimmter Subscriber sowohl Video- als auch Audiodaten empfängt.<br><br>Einheit: Bits pro Sekunde<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Bitrate über das konfigurierte Intervall. |

| Kennzahl | Dimension | Beschreibung |
| --- | --- | --- |
| `SubscribeBitrate` | `Stage,Participant,MediaType` | Filtert `SubscribeBitrate` nach Teilnehmer, für Subscriber, die auch Publisher sind. Beispiele stellen die Bitrate dar, mit der ein bestimmter Subscriber den jeweiligen `MediaType` empfängt. Beispiele werden nur ausgegeben, während der abonnierende Teilnehmer veröffentlicht.<br><br>Einheit: Bits pro Sekunde<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum – Durchschnittliche Anzahl, größte bzw. kleinste Anzahl der Bitrate über das konfigurierte Intervall. |
| `Subscribers` | `Stage` | Anzahl der Teilnehmer, die Subscriber der Stufe sind. Beachten Sie, dass Teilnehmer, die aktiv veröffentlichen und abonnieren, sowohl als Publisher als auch als Subscriber gezählt werden.<br><br>Einheit: Anzahl<br><br>Gültige Statistiken: Durchschnitt, Maximum, Minimum |

# IVS-Broadcast-SDK (Echtzeit-Streaming)

Das Amazon Interactive Video Services (IVS)-Broadcast-SDK ist für Entwickler gedacht, die Anwendungen mit Amazon IVS erstellen. Dieses SDK wurde entwickelt, um die Amazon-IVS-Architektur zu nutzen und bietet neben Amazon IVS kontinuierliche Verbesserungen und neue Funktionen. Als natives Broadcast-SDK wurde es entwickelt, um die Leistungsauswirkungen auf Ihre Anwendung und auf die Geräte, mit denen Ihre Benutzer auf Ihre Anwendung zugreifen, zu minimieren.

Beachten Sie, dass das Broadcast-SDK sowohl für das Senden als auch für das Empfangen von Videos verwendet wird. Sie verwenden also dasselbe SDK für Hosts und Zuschauer. Kein separates Player-SDK erforderlich.

Ihre Anwendung kann die wichtigsten Funktionen des Amazon-IVS-Broadcast-SDK nutzen:

- Hochqualitatives Streaming – Das Broadcast-SDK unterstützt qualitativ hochwertiges Streaming. Nehmen Sie Videos von Ihrer Kamera auf und kodieren Sie sie mit bis zu 720p.
- Automatische Bitratenanpassungen – Smartphone-Nutzer sind mobil, so dass sich ihre Netzwerkbedingungen im Laufe einer Sendung ändern können. Das Amazon-IVS-Broadcast-SDK passt die Videobitrate automatisch an sich ändernde Netzwerkbedingungen an.
- Hoch- und Quer-Support – Unabhängig davon, wie Ihre Benutzer ihre Geräte halten, wird das Image mit der rechten Seite nach oben und richtig skaliert angezeigt. Das Broadcast-SDK unterstützt sowohl die Leinwandgröße im Hoch- als auch im Querformat. Es verwaltet automatisch das Seitenverhältnis, wenn die Benutzer ihr Gerät von der konfigurierten Ausrichtung weg drehen.
- Sicheres Streaming – Die Übertragungen Ihrer Benutzer werden mit TLS verschlüsselt, sodass sie ihre Streams sicher halten können.
- Externe Audiogeräte – Das Amazon-IVS-Broadcast-SDK unterstützt externe Audiobuchse, USB und Bluetooth-SCO-Mikrofone.

# Plattform-Anforderungen

## Native Plattformen

| Plattform | Unterstützte Versionen |
|---|---|
| Android | 9.0 und höher – Hinweis: Kunden können mit Version 5.0 entwickeln, werden aber nicht in der Lage sein, die Echtzeit-Streaming-Funktion zu nutzen. |
| iOS | 14 und höher |

IVS unterstützt mindestens 4 Hauptversionen von iOS und 6 Hauptversionen von Android. Unsere aktuelle Versionsunterstützung kann über diese Mindestanforderungen hinausgehen. Kunden werden über SDK-Versionshinweise mindestens 3 Monate im Voraus benachrichtigt, wenn eine Hauptversion nicht mehr unterstützt wird.

## Desktop-Browser

| Browser | Unterstützte Plattformen | Unterstützte Versionen |
|---|---|---|
| Chrome | Windows, macOS | Zwei Hauptversionen (aktuelle und neueste Vorversion) |
| Firefox | Windows, macOS | Zwei Hauptversionen (aktuelle und neueste Vorversion) |
| Edge | Windows 8.1 und höher | Zwei Hauptversionen (aktuelle und neueste Vorversion)<br><br>Schließt Edge Legacy aus |
| Safari | macOS | Zwei Hauptversionen (aktuelle und neueste Vorversion) |

## Mobile Browser (iOS und Android)

| Browser | Unterstützte Plattformen | Unterstützte Versionen |
|---|---|---|
| Chrome | iOS, Android | Zwei Hauptversionen (aktuelle und neueste Vorversion) |
| Firefox | Android | Zwei Hauptversionen (aktuelle und neueste Vorversion) |
| Safari | iOS | Zwei Hauptversionen (aktuelle und neueste Vorversion) |

### Bekannte Beschränkungen

- Aufgrund von Problemen mit Videoartefakten und schwarzen Bildschirmen raten wir davon ab, auf allen mobilen Geräten mit vier oder mehr Teilnehmern gleichzeitig zu veröffentlichen/abonnieren. Wenn Sie mehr Teilnehmer benötigen, konfigurieren Sie die reine Audio-Veröffentlichung und -Abonnierung.
- Aus Gründen der Leistung und möglicher Abstürze raten wir davon ab, eine Stufe zusammenzustellen und an einen Kanal im Android Mobile Web zu übertragen. Wenn Broadcast-Funktionalität erforderlich ist, integrieren Sie das Android-Broadcast-SDK für IVS-Echtzeit-Streaming.

## Webansichten

Das Web-Broadcast-SDK bietet keine Unterstützung für Webviews oder webähnliche Umgebungen (TV, Konsolen usw.). Informationen zu mobilen Implementierungen finden Sie im Broadcast-SDK-Handbuch für Echtzeit-Streaming mit niedriger Latenz für Android und iOS.

## Erforderlicher Gerätezugriff

Das Broadcast-SDK erfordert Zugriff auf die Kameras und Mikrofone des Geräts, sowohl auf die im Gerät integrierten als auch auf die über Bluetooth, USB oder eine Audiobuchse angeschlossenen.

# Support

Hinweis: Das Broadcast-SDK wird ständig verbessert. Siehe Versionshinweise zu Amazon IVS für verfügbare Versionen und behobene Probleme. Aktualisieren Sie gegebenenfalls Ihre Version des Broadcast-SDK, bevor Sie sich an den Support wenden und prüfen Sie, ob das Problem dadurch behoben wird.

## Versioning

Die Amazon-IVS-Broadcast-SDKs verwenden Semantisches Versioning.

Nehmen Sie für diese Diskussion an:

- Die neueste Version ist 4.1.3.
- Die neueste Version der vorherigen Hauptversion ist 3.2.4.
- Die neueste Version 1.x ist 1.5.6.

Rückwärtskompatible neue Funktionen werden als Nebenversionen der neuesten Version hinzugefügt. In diesem Fall wird der nächste Satz neuer Funktionen als Version 4.2.0 hinzugefügt.

Rückwärtskompatible, kleinere Fehlerbehebungen werden als Patch-Releases der neuesten Version hinzugefügt. Hier wird der nächste Satz von kleineren Fehlerbehebungen als Version 4.1.4 hinzugefügt.

Rückwärtskompatible, große Fehlerbehebungen werden unterschiedlich behandelt; diese werden zu mehreren Versionen hinzugefügt:

- Patch-Version der neuesten Version. Hier ist das Version 4.1.4.
- Patch-Version der vorherigen Nebenversion. Hier ist das Version 3.2.5.
- Patch-Version der neuesten Version 1.x. Hier ist das Version 1.5.7.

Wichtige Fehlerbehebungen werden vom Amazon IVS-Produktteam definiert. Typische Beispiele sind kritische Sicherheitsupdates und ausgewählte andere Korrekturen, die für Kunden erforderlich sind.

Hinweis: In den obigen Beispielen werden freigegebene Versionen inkrementiert, ohne dass Zahlen übersprungen werden (z. B. von 4.1.3 auf 4.1.4). In Wirklichkeit können eine oder mehrere Patch-Nummern intern bleiben und nicht veröffentlicht werden, so dass die freigegebene Version von 4.1.3 auf, sagen wir, 4.1.6 steigen könnte.

# IVS-Broadcast-SDK: Web-Handbuch (Echtzeit-Streaming)

Das Web-Broadcast-SDK von IVS gibt Entwicklern die Werkzeuge an die Hand, um interaktive Echtzeit-Erlebnisse im Web zu schaffen. Dieses SDK ist für Entwickler gedacht, die Webanwendungen mit Amazon IVS erstellen.

Das Web-Broadcast-SDK ermöglicht es den Teilnehmern, Videos zu senden und zu empfangen. Das SDK unterstützt die folgenden Vorgänge:

- Einer Bühne beitreten
- Medien für andere Teilnehmer auf der Bühne veröffentlichen
- Medien anderer Teilnehmer auf der Bühne abonnieren
- Auf der Bühne veröffentlichte Videos und Audios verwalten und überwachen
- WebRTC-Statistiken für jede Peer-Verbindung beziehen
- Alle Operationen aus dem Web-Broadcast-SDK für IVS-Streaming mit niedriger Latenz

Neueste Version des Web-Broadcast-SDK: 1.8.0 (Versionshinweise)

Referenzdokumentation: Informationen zu den wichtigsten Methoden, die im Amazon IVS Web Broadcast SDK verfügbar sind, finden Sie unter https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference . Stellen Sie sicher, dass die neueste Version des SDK ausgewählt ist.

Beispielcode: Die folgenden Beispiele sind ein guter Ausgangspunkt, um schnell mit dem SDK loszulegen:

- HTML und JavaScript
- React

Plattformanforderungen: Eine Liste der unterstützten Plattformen finden Sie unter Amazon IVS Broadcast SDK.

## Erste Schritte

### Importe

Die Bausteine für Echtzeit befinden sich in einem anderen Namespace als die Root-Broadcasting-Module.

Verwenden eines Skript-Tags

Die in den folgenden Beispielen definierten Klassen und Enums lassen sich mit denselben Skriptimporten im globalen Objekt `IVSBroadcastClient` finden:

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

Verwenden von npm

Die Klassen, Enums und Typen können auch aus dem Paketmodul importiert werden:

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

## Berechtigungen anfordern

Ihre App muss die Berechtigung für den Zugriff auf die Kamera und das Mikrofon des Benutzers anfordern und muss über HTTPS bereitgestellt werden. (Das gilt nicht nur für Amazon IVS, sondern für alle Websites, die Zugriff auf Kameras und Mikrofone benötigen.)

Die folgende Beispielfunktion zeigt, wie Sie Berechtigungen für Audio- und Videogeräte anfordern und erfassen können:

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

Weitere Informationen finden Sie in der Berechtigungs-API und MediaDevices.getUserMedia().

### Auflisten der verfügbaren Geräte

Um zu sehen, welche Geräte erfasst werden können, fragen Sie die Methode MediaDevices.enumerateDevices () des Browsers ab:

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

### Abrufen eines MediaStream von einem Gerät

Nachdem Sie die Liste der verfügbaren Geräte erfasst haben, können Sie einen Stream von einer beliebigen Anzahl von Geräten abrufen. Sie können zum Beispiel mit der Methode `getUserMedia()` einen Stream von einer Kamera abrufen.

Wenn Sie angeben möchten, von welchem Gerät der Stream erfasst werden soll, können Sie die `deviceId` im Bereich `audio` oder `video` der Medieneinschränkungen explizit festlegen. Alternativ können Sie die `deviceId` weglassen und Benutzer ihre Geräte über die Eingabeaufforderung des Browsers auswählen lassen.

Zudem können Sie mithilfe der Einschränkungen `width` und `height` eine ideale Kameraauflösung angeben. (Mehr über diese Einschränkungen erfahren Sie hier.) Das SDK wendet automatisch die Einschränkungen für die Breite und Höhe an, die Ihrer maximalen Übertragungsauflösung entsprechen. Es empfiehlt sich jedoch, diese auch selbst anzuwenden, damit das Seitenverhältnis der Quelle nicht geändert wird, nachdem Sie sie dem SDK hinzugefügt haben.

Stellen Sie für Echtzeit-Streaming sicher, dass die Medien auf eine Auflösung von 720p beschränkt sind. Insbesondere dürfen Ihre `getDisplayMedia` - `getUserMedia` und -Einschränkungswerte für Breite und Höhe 921600 (1280*720) nicht überschreiten, wenn sie miteinander multipliziert werden.

```
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 },
});
```

# Publishing und Subscribing

## Konzepte

Drei Kernkonzepte liegen der Echtzeit-Funktionalität zugrunde: Stage, Strategie und Ereignisse. Das Designziel besteht in der Minimierung der Menge an clientseitiger Logik, die für die Entwicklung eines funktionierenden Produkts erforderlich ist.

### Stufe

Die Klasse `Stage` ist der Hauptinteraktionspunkt zwischen der Hostanwendung und dem SDK. Sie stellt die Bühne selbst dar und dient dazu, der Bühne beizutreten und sie zu verlassen. Für das Erstellen einer Bühne und das Beitreten ist eine gültige, noch nicht abgelaufene Token-Zeichenfolge aus der Steuerebene erforderlich (dargestellt als `token`). Einer Bühne beizutreten und sie zu verlassen, ist ganz einfach:

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

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

```
}

stage.leave();
```

### Strategie

Über die Schnittstelle `StageStrategy` kann die Hostanwendung dem SDK den gewünschten Status der Bühne mitteilen. Drei Funktionen müssen implementiert werden: `shouldSubscribeToParticipant`, `shouldPublishParticipant` und `stageStreamsToPublish`. Alle werden im Folgenden behandelt.

Um eine definierte Strategie zu verwenden, übergeben Sie sie an den `Stage`-Konstruktor. Im Folgenden finden Sie ein vollständiges Beispiel für eine Anwendung, die mithilfe einer Strategie die Webcam eines Teilnehmers auf der Bühne veröffentlicht und alle Teilnehmer abonniert. Der Zweck der einzelnen erforderlichen Strategiefunktionen wird in den folgenden Abschnitten ausführlich erläutert.

```
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();
```

Abonnieren von Teilnehmern

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

Wenn ein Remote-Teilnehmer der Bühne beitritt, fragt das SDK die Hostanwendung nach dessen gewünschtem Abonnementstatus. Die Optionen lauten `NONE`, `AUDIO_ONLY` und `AUDIO_VIDEO`. Wenn ein Wert für diese Funktion zurückgegeben wird, muss sich die Hostanwendung nicht um den Veröffentlichungs-, den aktuellen Abonnement- oder den Verbindungsstatus des Bühne kümmern. Bei Rückgabe von `AUDIO_VIDEO` wartet das SDK mit dem Abonnieren, bis der Remote-Teilnehmer etwas veröffentlicht. Außerdem aktualisiert es die Hostanwendung, indem es während des gesamten Prozesses Ereignisse ausgibt.

Hier folgt ein Beispiel für eine Implementierung:

```
const strategy = {

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

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

Hierbei handelt es sich um die vollständige Implementierung dieser Funktion für eine Hostanwendung, bei der sich alle Teilnehmer stets gegenseitig sehen sollen; z. B. eine Video-Chat-Anwendung.

Weitergehende Implementierungen sind ebenfalls möglich. Nutzen Sie die Eigenschaft `userInfo` für `ParticipantInfo`, um Teilnehmer anhand der vom Server bereitgestellten Attribute selektiv zu abonnieren:

```
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
}
```

Hiermit kann eine Bühne erstellt werden, auf der Moderatoren alle Gäste überwachen können, ohne selbst gesehen oder gehört zu werden. Die Hostanwendung könnte eine zusätzliche Geschäftslogik nutzen, damit Moderatoren sich gegenseitig sehen können, für Gäste aber unsichtbar bleiben.

Veröffentlichen

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

Sobald die Verbindung zur Bühne hergestellt ist, überprüft das SDK per Anfrage an die Hostanwendung, ob ein bestimmter Teilnehmer etwas veröffentlichen soll. Dies wird nur bei lokalen Teilnehmern aufgerufen, die auf Grundlage des bereitgestellten Tokens zur Veröffentlichung berechtigt sind.

Hier folgt ein Beispiel für eine Implementierung:

```
const strategy = {

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

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

Sie ist für eine normale Video-Chat-Anwendung gedacht, bei der Benutzer immer etwas veröffentlichen möchten. Sie können die Audio- und Videowiedergabe stummschalten und die Stummschaltung aufheben, um umgehend ausgeblendet oder gesehen/gehört zu werden. (Sie können auch „Veröffentlichen/Veröffentlichung aufheben“ verwenden, was aber viel langsamer ist. „Stummschalten/Stummschalten aufheben“ ist für Anwendungsfälle vorzuziehen, in denen eine häufige Änderung der Sichtbarkeit wünschenswert ist.)

Auswählen von Streams zur Veröffentlichung

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

Beim Veröffentlichen wird hiermit bestimmt, welche Audio- und Videostreams veröffentlicht werden sollen. Dieser Punkt wird später unter Veröffentlichen eines Medienstreams ausführlicher behandelt.

Aktualisieren der Strategie

Die Strategie soll dynamisch sein: Die von einer der oben genannten Funktionen zurückgegebenen Werte lassen sich jederzeit ändern. Wenn die Hostanwendung beispielsweise erst veröffentlichen soll, wenn der Endbenutzer auf eine Schaltfläche tippt, können Sie eine Variable aus `shouldPublishParticipant` zurückgeben (zum Beispiel `hasUserTappedPublishButton`). Wenn sich diese Variable aufgrund einer Interaktion des Endbenutzers ändert, signalisieren Sie dem SDK per Aufruf von `stage.refreshStrategy()`, dass es die Strategie nach den neuesten Werten abfragen und nur Dinge anwenden soll, die sich geändert haben. Wenn das SDK feststellt, dass sich der Wert `shouldPublishParticipant` geändert hat, startet es den Veröffentlichungsprozess. Wenn alle Funktionen bei einer SDK-Abfrage den gleichen Wert zurückgeben wie zuvor, wird die Bühne mit dem Aufruf von `refreshStrategy` nicht geändert.

Ändert sich der Rückgabewert von `shouldSubscribeToParticipant` von `AUDIO_VIDEO` in `AUDIO_ONLY`, wird der Videostream für alle Teilnehmer mit geänderten Rückgabewerten entfernt, sofern zuvor ein Videostream vorhanden war.

Im Allgemeinen nutzt die Bühne die Strategie, um den Unterschied zwischen der vorherigen und der aktuellen Strategie am effizientesten anzuwenden. Dabei muss sich die Hostanwendung nicht um die ganzen Status kümmern, die für eine ordnungsgemäße Verwaltung erforderlich sind. Stellen Sie sich den Aufruf von `stage.refreshStrategy()` daher als einen ressourcenschonenden Vorgang vor, da nur bei einer Änderung der Strategie etwas unternommen wird.

Ereignisse

Eine `Stage`-Instance ist ein Ereignis-Emitter. Mit `stage.on()` wird der Hostanwendung der Status der Bühne mitgeteilt. Aktualisierungen in der Benutzeroberfläche der Hostanwendung können in der Regel vollständig durch die Ereignisse unterstützt werden. Folgende Ereignisse werden unterstützt:

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

Für die meisten dieser Ereignisse wird die entsprechende `ParticipantInfo` bereitgestellt.

Es wird nicht erwartet, dass sich die von den Ereignissen bereitgestellten Informationen auf die Rückgabewerte der Strategie auswirken. Es wird beispielsweise nicht erwartet, dass sich der Rückgabewert von `shouldSubscribeToParticipant` beim Aufruf von `STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` ändert. Wenn die Hostanwendung einen bestimmten Teilnehmer abonnieren möchte, muss sie unabhängig von dessen Veröffentlichungsstatus den gewünschten Abonnementtyp zurückgeben. Das SDK muss dafür sorgen, dass entsprechend dem Status der Bühne und dem gewünschten Status der Strategie zum richtigen Zeitpunkt gehandelt wird.

## Veröffentlichen eines Medienstreams

Lokale Geräte wie Mikrofone und Kameras werden mit den gleichen Schritten abgerufen, die oben unter Abrufen eines MediaStream von einem Gerät beschrieben sind. In dem Beispiel erstellen wir mit `MediaStream` eine Liste von `LocalStageStream`-Objekten, die für die Veröffentlichung durch das SDK verwendet werden:

```
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
    }
}
```

## Veröffentlichen einer Bildschirmfreigabe

Häufig müssen Anwendungen zusätzlich zur Webkamera des Benutzers eine Bildschirmfreigabe veröffentlichen. Dazu muss eine zusätzliche `Stage` mit einem eigenen eindeutigen Token erstellt werden.

```
// 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();
```

## Anzeigen und Entfernen von Teilnehmern

Nach Abschluss von Abonnements erhalten Sie über das Ereignis `STAGE_PARTICIPANT_STREAMS_ADDED` eine Reihe von `StageStream`-Objekten. Zudem stellt das Ereignis Teilnehmerinformationen bereit, die Ihnen beim Anzeigen von Medienstreams helfen:

```
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));
})
```

Wenn ein Teilnehmer die Veröffentlichung beendet oder dessen Abonnement eines Streams beendet wird, wird die Funktion `STAGE_PARTICIPANT_STREAMS_REMOVED` mit den Streams aufgerufen, die entfernt wurden. Hostanwendungen sollten dies als Signal nutzen, um den Videostream des Teilnehmers aus dem DOM zu entfernen.

`STAGE_PARTICIPANT_STREAMS_REMOVED` wird für alle Szenarien aufgerufen, in denen ein Stream entfernt werden könnte, darunter:

- Der Remote-Teilnehmer beendet die Veröffentlichung.
- Ein lokales Gerät beendet das Abonnement oder ändert das Abonnement von `AUDIO_VIDEO` in `AUDIO_ONLY`.
- Der Remote-Teilnehmer verlässt die Bühne.

- Der lokale Teilnehmer verlässt die Bühne.

Da `STAGE_PARTICIPANT_STREAMS_REMOVED` bei allen Szenarien aufgerufen wird, ist keine benutzerdefinierte Geschäftslogik erforderlich, um Teilnehmer beim remoten oder lokalen Verlassen aus der Benutzeroberfläche zu entfernen.

### Stummschalten von Medienstreams und Aufheben der Stummschaltung

`LocalStageStream`-Objekte verfügen über eine `setMuted`-Funktion, die das Stummschalten des Streams steuert. Diese Funktion kann für den Stream aufgerufen werden, bevor oder nachdem er von der Strategiefunktion `stageStreamsToPublish` zurückgegeben wird.

Wichtig: Wenn nach einem Aufruf von `refreshStrategy` eine neue `LocalStageStream`-Objekt-Instance von `stageStreamsToPublish` zurückgegeben wird, wird der Stummschaltungsstatus des neuen Streamobjekts auf die Bühne angewendet. Seien Sie vorsichtig beim Erstellen neuer `LocalStageStream`-Instances, um sicherzustellen, dass der erwartete Stummschaltungsstatus beibehalten wird.

### Überwachen des Medien-Stummschaltungsstatus von Remote-Teilnehmern

Wenn Teilnehmer den Stummschaltungsstatus ihres Videos oder Audios ändern, wird das Ereignis `STAGE_STREAM_MUTE_CHANGED` mit einer Liste der Streams ausgelöst, die sich geändert haben. Verwenden Sie die Eigenschaft `isMuted` für `StageStream`, um die Benutzeroberfläche entsprechend zu aktualisieren:

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

Sie können sich auch StageParticipantInfo Zustandsinformationen darüber ansehen, ob Audio oder Video stummgeschaltet sind:

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

## Abrufen von WebRTC-Statistiken

Um die neuesten WebRTC-Statistiken für einen veröffentlichten oder abonnierten Stream abzurufen, verwenden Sie `getStats` für `StageStream`. Hierbei handelt es sich um eine asynchrone Methode, mit der Sie Statistiken entweder über await oder durch Verkettung eines Promise abrufen können. Das Ergebnis ist ein `RTCStatsReport`, ein Wörterbuch, das alle Standardstatistiken enthält.

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

## Optimieren von Medien

Für eine optimale Leistung wird empfohlen, Aufrufe von `getUserMedia` und `getDisplayMedia` entsprechend den folgenden Einschränkungen zu begrenzen:

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

Sie können die Medien durch zusätzliche Optionen, die an den `LocalStageStream`-Konstruktor übergeben werden, weiter einschränken:

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

Im obigen Code:

- `minBitrate` legt eine Mindestbitrate fest, die der Browser voraussichtlich verwenden sollte. Ein Videostream mit geringer Komplexität kann jedoch dazu führen, dass der Encoder diese Bitrate unterschreitet.
- `maxBitrate` legt eine maximale Bitrate fest, von der erwartet werden sollte, dass sie vom Browser für diesen Stream nicht überschritten wird.
- `maxFramerate` legt eine maximale Framerate fest, von der erwartet werden sollte, dass sie vom Browser für diesen Stream nicht überschritten wird.
- Die Option `simulcast` ist nur in Chromium-basierten Browsern verwendbar. Sie ermöglicht das Senden von drei Wiedergabeebenen des Streams.
  - Auf diese Weise kann der Server anhand ihrer Netzwerkbeschränkungen auswählen, welche Wiedergabeversion an andere Teilnehmer gesendet werden soll.
  - Wenn `simulcast` zusammen mit einem `maxBitrate` und/oder `maxFramerate` Wert angegeben wird, wird erwartet, dass die höchste Wiedergabe-Ebene unter Berücksichtigung dieser Werte konfiguriert wird, vorausgesetzt, `maxBitrate` unterschreitet nicht die Standardeinstellung der zweithöchsten Ebene des internen SDK-Standardwerts `maxBitrate` von 900 kbps.
  - Wenn `maxBitrate` im Vergleich zum Standardwert der zweithöchsten Ebene als zu niedrig angegeben wird, wird `simulcast` deaktiviert.
  - `simulcast` kann nicht ein- und ausgeschaltet werden, ohne die Medien erneut zu veröffentlichen, indem `shouldPublishParticipant` `false` zurückgibt, `refreshStrategy` aufruft, `shouldPublishParticipant` `true` zurückgibt, und `refreshStrategy` wieder aufruft.

## Abrufen von Teilnehmerattributen

Wenn Sie Attribute in der Endpunktanfrage `CreateParticipantToken` angeben, können Sie die Attribute in den Eigenschaften von `StageParticipantInfo` einsehen:

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

## Umgang mit Netzwerkproblemen

Bei Unterbrechung der Netzwerkverbindung des lokalen Geräts versucht das SDK intern, die Verbindung ohne Benutzeraktion wiederherzustellen. In einigen Fällen ist das SDK nicht erfolgreich, weshalb eine Benutzeraktion erforderlich ist.

Generell kann der Status der Bühne über das Ereignis `STAGE_CONNECTION_STATE_CHANGED` gesteuert werden:

```
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;
})
```

Im Allgemeinen deutet das Auftreten von Fehlern nach dem erfolgreichen Beitritt zu einer Bühne darauf hin, dass die Verbindung unterbrochen wurde und das SDK beim Wiederherstellen einer Verbindung nicht erfolgreich war. Erstellen Sie ein neues `Stage`-Objekt und versuchen Sie, der Bühne beizutreten, wenn sich die Netzwerkbedingungen verbessern.

## Übertragung der Stage auf einen IVS-Kanal

Zum Übertragen einer Bühne erstellen Sie eine separate `IVSBroadcastClient`-Sitzung und folgen Sie dann den oben beschriebenen üblichen Anweisungen für die Übertragung mit dem SDK. Mithilfe der Liste der über `STAGE_PARTICIPANT_STREAMS_ADDED` offengelegten `StageStream` können die Medienstreams der Teilnehmer abgerufen werden, die wie folgt auf die Zusammensetzung der übertragenen Streams angewendet werden können:

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

Optional können Sie eine Stage zusammenstellen und sie auf einen IVS-Kanal mit niedriger Latenz übertragen, um ein größeres Publikum zu erreichen. Sehen Sie Aktivierung mehrerer Hosts in einem Amazon-IVS-Stream im Benutzerhandbuch für IVS-Streaming mit niedriger Latenz.

## Bekannte Probleme und Problemumgehungen

- Wenn Browser-Tabs oder Browser ohne Aufruf von `stage.leave()` geschlossen werden, können Benutzer noch bis zu 10 Sekunden lang mit einem eingefrorenen Frame oder einem schwarzen Bildschirm in der Sitzung zu sehen sein.

  Problemumgehung: Keine.
- Safari-Sitzungen werden für Benutzer, die nach Beginn einer Sitzung beitreten, mitunter mit einem schwarzen Bildschirm angezeigt.

  Problemumgehung: Aktualisieren Sie den Browser und stellen Sie die Verbindung zur Sitzung erneut her.
- Safari stellt Sitzungen bei einem Netzwerkwechsel nicht ordnungsgemäß wieder her.

  Problemumgehung: Aktualisieren Sie den Browser und stellen Sie die Verbindung zur Sitzung erneut her.

- Die Entwicklerkonsole wiederholt den Fehler `Error: UnintentionalError at StageSocket.onClose`.

  Problemumgehung: Pro Teilnehmertoken kann nur eine Bühne erstellt werden. Dieser Fehler tritt auf, wenn mehr als eine `Stage`-Instance mit demselben Teilnehmertoken erstellt wird, unabhängig davon, ob sich die Instance auf einem oder mehreren Geräten befindet.

- Möglicherweise haben Sie Probleme, einen `-StageParticipantPublishState.PUBLISHED`Zustand beizubehalten, und können beim Abhören des `StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` Ereignisses wiederholte `StageParticipantPublishState.ATTEMPTING_PUBLISH` Zustände erhalten.

  Problemumgehung: Beschränken Sie die Videoauflösung beim Aufrufen von `getUserMedia` oder auf 720p`getDisplayMedia`. Insbesondere dürfen Ihre `getDisplayMedia` - `getUserMedia` und -Einschränkungswerte für Breite und Höhe 921600 (1280*720) nicht überschreiten, wenn sie miteinander multipliziert werden.

## Einschränkungen von Safari

- Wenn bei einer entsprechenden Aufforderung die Erteilung einer Berechtigung verweigert wird, muss die Berechtigung in den Einstellungen auf der Safari-Website auf Betriebssystemebene zurückgesetzt werden.
- Safari erkennt nicht alle Geräte nativ so effektiv wie Firefox oder Chrome. OBS Virtual Camera wird beispielsweise nicht erkannt.

## Einschränkungen von Firefox

- Damit Firefox den Bildschirm freigeben kann, müssen Systemberechtigungen aktiviert sein. Nach der Aktivierung muss der Benutzer Firefox neu starten, damit es ordnungsgemäß funktioniert. Andernfalls löst der Browser eine NotFoundError Ausnahme aus, wenn Berechtigungen als blockiert wahrgenommen werden.
- Die Methode `getCapabilities` fehlt. Das bedeutet, dass Benutzer die Auflösung oder das Seitenverhältnis der Medienspur nicht abrufen können. Weitere Informationen finden Sie in diesem Bugzilla-Thread.
- Es fehlen mehrere `AudioContext`-Eigenschaften, z. B. die Latenz und die Kanalanzahl. Dies könnte für erfahrene Benutzer, die die Audiospuren bearbeiten möchten, ein Problem darstellen.

- Kamera-Feeds von `getUserMedia` sind unter macOS auf ein Seitenverhältnis von 4:3 beschränkt. Weitere Informationen finden Sie im Bugzilla-Thread 1 und im Bugzilla-Thread 2.

- Die Audioerfassung wird mit `getDisplayMedia` nicht unterstützt. Weitere Informationen finden Sie in diesem Bugzilla-Thread.
- Die Framerate bei der Bildschirmerfassung ist suboptimal (ungefähr 15 Bilder pro Sekunde?). Weitere Informationen finden Sie in diesem Bugzilla-Thread.

## Einschränkungen im mobilen Web

- getDisplayMedia Die Bildschirmfreigabe wird auf Mobilgeräten nicht unterstützt.

  Problemumgehung: Keine.
- Beim Schließen eines Browsers dauert es 15 bis 30 Sekunden, bis der Teilnehmer den Browser verlässt, ohne `leave()` aufzurufen.

  Problemumgehung: Fügen Sie eine Benutzeroberfläche hinzu, die Benutzer dazu ermutigt, die Verbindung ordnungsgemäß zu trennen.
- Die Hintergrund-App führt dazu, dass die Veröffentlichung von Videos beendet wird.

  Problemumgehung: Zeigen Sie ein UI-Slate an, wenn der Publisher angehalten ist.
- Nach dem Aufheben der Stummschaltung einer Kamera auf Android-Geräten sinkt die Video-Framerate für etwa 5 Sekunden.

  Problemumgehung: Keine.
- Der Video-Feed wird bei der Rotation für iOS 16.0 gestreckt.

  Problemumgehung: Zeigen Sie eine Benutzeroberfläche an, die dieses bekannte Betriebssystemproblem beschreibt.
- Beim Wechseln des Audio-Eingabegeräts wird automatisch auch das Audio-Ausgabegerät umgeschaltet.

  Problemumgehung: Keine.
- Wenn der Browser im Hintergrund abgelegt wird, wird der Veröffentlichungs-Stream schwarz und erzeugt nur Audio.

  Problemumgehung: Keine. Dies liegt aus Sicherheitsgründen vor.

# Fehlerbehandlung

Dieser Abschnitt gibt einen Überblick über die Fehlerbedingungen, wie das Web-Broadcast-SDK sie an die Anwendung meldet und was eine Anwendung tun sollte, wenn diese Fehler auftreten. Es gibt vier Kategorien von Fehlern:

```
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
  }
});
```

## Fehler bei der Instanziierung der Phase

Bei der Phaseninstanziierung werden Tokens nicht per Fernzugriff validiert, es wird jedoch nach einigen grundlegenden Token-Problemen gesucht, die auf der Client-Seite überprüft werden können. Infolgedessen kann das SDK einen Fehler ausgeben.

### Fehlerhaftes Teilnehmer-Token

Dies tritt auf, wenn das Phasen-Token falsch formatiert ist. Beim Instanziieren einer Phase gibt das SDK einen Fehler mit der folgenden Meldung aus: „Fehler beim Parsen des Stage-Token“.

Aktion: Erstellen Sie ein gültiges Token und versuchen Sie erneut, es zu instanziieren.

## Fehler beim Beitreten zu einer Phase

Dies sind die Fehler, die auftreten können, wenn Sie zum ersten Mal versuchen, einer Phase beizutreten.

Phase wurde gelöscht

Dies tritt auf, wenn Sie einer Phase beitreten (die mit einem Token verknüpft ist), die gelöscht wurde. Die `join` SDK-Methode gibt einen Fehler mit der folgenden Meldung aus: „InitialConnectTimedOut nach 10 Sekunden“.

Aktion: Erstellen Sie ein gültiges Token mit einer neuen Phase und versuchen Sie erneut beizutreten.

Abgelaufenes Teilnehmer-Token

Dies tritt auf, wenn das Token abgelaufen ist. Die SDK-Methode `join` gibt einen Fehler mit der folgenden Meldung aus: „Das Token ist abgelaufen und nicht mehr gültig.“

Aktion: Erstellen Sie ein neues Token und versuchen Sie erneut beizutreten.

Ungültiges oder widerrufenes Teilnehmer-Token

Dies tritt auf, wenn das Token ungültig ist oder widerrufen/die Verbindung unterbrochen wurde. Die `join` SDK-Methode gibt einen Fehler mit der folgenden Meldung aus: „InitialConnectTimedOut nach 10 Sekunden“.

Aktion: Erstellen Sie ein neues Token und versuchen Sie erneut beizutreten.

Token getrennt

Dies tritt auf, wenn das Phasen-Token nicht fehlerhaft formatiert ist, sondern vom Stages-Server zurückgewiesen wird. Die `join` SDK-Methode gibt einen Fehler mit der folgenden Meldung aus: „InitialConnectTimedOut nach 10 Sekunden“.

Aktion: Erstellen Sie ein gültiges Token und versuchen Sie erneut beizutreten.

Netzwerkfehler beim ersten Beitritt

Dies tritt auf, wenn das SDK den Stages-Server nicht kontaktieren kann, um eine Verbindung herzustellen. Die `join` SDK-Methode gibt einen Fehler mit der folgenden Meldung aus: „InitialConnectTimedOut nach 10 Sekunden“.

Handlung: Warten Sie, bis die Konnektivität des Geräts wiederhergestellt ist, und versuchen Sie erneut, eine Verbindung herzustellen.

Netzwerkfehler, wenn bereits eine Verbindung hergestellt wurde

Wenn die Netzwerkverbindung des Geräts ausfällt, verliert das SDK möglicherweise die Verbindung zu den Phasen-Servern. Möglicherweise werden in der Konsole Fehler angezeigt, da das SDK die Back-End-Dienste nicht mehr erreichen kann. POSTs auf https://broadcast.stats.live-video.net schlagen fehl.

Wenn Sie etwas veröffentlichen und/oder abonnieren, werden in der Konsole Fehler angezeigt, die sich auf Versuche beziehen, etwas zu veröffentlichen/zu abonnieren.

Intern versucht das SDK, die Verbindung mithilfe einer exponentiellen Backoff-Strategie wiederherzustellen.

Aktion: Warten Sie, bis die Konnektivität des Geräts wiederhergestellt ist. Wenn Sie etwas veröffentlichen oder abonnieren, aktualisieren Sie die Strategie, um sicherzustellen, dass Ihre Medienstreams erneut veröffentlicht werden.

## Fehler beim Veröffentlichen und Abonnieren

Fehler bei der Veröffentlichung: States veröffentlichen

Das SDK meldet `ERRORED`, wenn eine Veröffentlichung fehlschlägt. Dies kann auf Netzwerkbedingungen zurückzuführen sein oder wenn eine Phase für Publisher ausgelastet ist.

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

Maßnahme: Aktualisieren Sie die Strategie, um zu versuchen, Ihre Medienstreams erneut zu veröffentlichen.

Fehler beim Abonnieren

Das SDK meldet `ERRORED`, wenn ein Abonnement fehlschlägt. Dies kann auf Netzwerkbedingungen zurückzuführen sein oder wenn eine Phase für Abonnenten ausgelastet ist.

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

Maßnahme: Aktualisieren Sie die Strategie, um ein neues Abonnement auszuprobieren.

# IVS-Broadcast-SDK: Android-Handbuch (Echtzeit-Streaming)

Das Android-Broadcast-SDK für IVS-Echtzeit-Streaming ermöglicht es den Teilnehmern, Videos auf Android zu senden und zu empfangen.

Das Paket `com.amazonaws.ivs.broadcast` implementiert die in diesem Dokument beschriebene Schnittstelle. Das SDK unterstützt die folgenden Vorgänge:

- Einer Bühne beitreten
- Medien für andere Teilnehmer auf der Bühne veröffentlichen
- Medien anderer Teilnehmer auf der Bühne abonnieren
- Auf der Bühne veröffentlichte Videos und Audios verwalten und überwachen
- WebRTC-Statistiken für jede Peer-Verbindung beziehen
- Alle Vorgänge aus dem IVS-Streaming-SDK für Android-Übertragungen mit niedriger Latenz

Neueste Version des Android-Broadcast-SDK: 1.14.1 (Versionshinweise )

Referenzdokumentation: Informationen zu den wichtigsten Methoden, die im Amazon IVS Android Broadcast SDK verfügbar sind, finden Sie in der Referenzdokumentation unter https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/android/.

Beispielcode: Siehe das Android-Beispiel-Repository auf GitHub: https://github.com/aws-samples/amazon-ivs-broadcast-android-sample .

Anforderungen an die Plattform: Android 9.0 und höher.

# Erste Schritte

## Installieren Sie die Bibliothek

Wenn Sie der Android-Entwicklungsumgebung die Amazon-IVS-Android-Broadcast-Bibliothek hinzufügen möchten, fügen Sie die Bibliothek der `build.gradle` – wie hier gezeigt – (für die neueste Version des Amazon-IVS-Broadcast-SDK) zu Ihren Modulen hinzu:

```
repositories {
    mavenCentral()
}

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

Fügen Sie dem Manifest die folgende Berechtigung hinzu, damit das SDK die Freisprecheinrichtung aktivieren und deaktivieren kann:

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

Um das SDK manuell zu installieren, laden Sie alternativ die neueste Version von diesem Speicherort herunter:

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

Laden Sie unbedingt die `aar` mit `-stages` angehängt herunter.

## Berechtigungen anfordern

Ihre App muss die Berechtigung für den Zugriff auf die Kamera und das Mikrofon des Benutzers anfordern. (Dies ist nicht spezifisch für Amazon IVS; es ist für alle Anwendungen erforderlich, die Zugriff auf Kameras und Mikrofone benötigen.)

Hier prüfen wir, ob der Benutzer bereits Berechtigungen erteilt hat und fragen, wenn nicht, nach ihnen:

```
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;
    }
}
```

Hier erhalten wir die Antwort des Benutzers:

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

# Publishing und Subscribing

## Konzepte

Drei Kernkonzepte liegen der Echtzeit-Funktionalität zugrunde: Stage, Strategie und Renderer. Das Designziel besteht in der Minimierung der Menge an clientseitiger Logik, die für die Entwicklung eines funktionierenden Produkts erforderlich ist.

### Stufe

Die Klasse `Stage` ist der Hauptinteraktionspunkt zwischen der Hostanwendung und dem SDK. Sie stellt die Bühne selbst dar und dient dazu, der Bühne beizutreten und sie zu verlassen. Für das Erstellen einer Bühne und das Beitreten ist eine gültige, noch nicht abgelaufene Token-Zeichenfolge aus der Steuerebene erforderlich (dargestellt als `token`). Einer Bühne beizutreten und sie zu verlassen, ist ganz einfach.

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

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

stage.leave();
```

In der Klasse `Stage` erfolgt auch das Anhängen des `StageRenderer`:

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

Strategie

Über die Schnittstelle `Stage.Strategy` kann die Hostanwendung dem SDK den gewünschten Status der Bühne mitteilen. Drei Funktionen müssen implementiert werden: `shouldSubscribeToParticipant`, `shouldPublishFromParticipant` und `stageStreamsToPublishForParticipant`. Alle werden im Folgenden behandelt.

Abonnieren von Teilnehmern

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

Wenn ein Remote-Teilnehmer der Bühne beitritt, fragt das SDK die Hostanwendung nach dessen gewünschtem Abonnementstatus. Die Optionen lauten `NONE`, `AUDIO_ONLY` und `AUDIO_VIDEO`. Wenn ein Wert für diese Funktion zurückgegeben wird, muss sich die Hostanwendung nicht um den Veröffentlichungs-, den aktuellen Abonnement- oder den Verbindungsstatus des Bühne kümmern. Bei Rückgabe von `AUDIO_VIDEO` wartet das SDK mit dem Abonnieren, bis der Remote-Teilnehmer etwas veröffentlicht. Außerdem aktualisiert das SDK die Hostanwendung während des gesamten Prozesses über den Renderer.

Hier folgt ein Beispiel für eine Implementierung:

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

```
}
```

Hierbei handelt es sich um die vollständige Implementierung dieser Funktion für eine Hostanwendung, bei der sich alle Teilnehmer stets gegenseitig sehen sollen; z. B. eine Video-Chat-Anwendung.

Weitergehende Implementierungen sind ebenfalls möglich. Nutzen Sie die Eigenschaft `userInfo` für `ParticipantInfo`, um Teilnehmer anhand der vom Server bereitgestellten Attribute selektiv zu abonnieren:

```
@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;
 }
}
```

Hiermit kann eine Bühne erstellt werden, auf der Moderatoren alle Gäste überwachen können, ohne selbst gesehen oder gehört zu werden. Die Hostanwendung könnte eine zusätzliche Geschäftslogik nutzen, damit Moderatoren sich gegenseitig sehen können, für Gäste aber unsichtbar bleiben.

Veröffentlichen

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

Sobald die Verbindung zur Bühne hergestellt ist, überprüft das SDK per Anfrage an die Hostanwendung, ob ein bestimmter Teilnehmer etwas veröffentlichen soll. Dies wird nur bei lokalen Teilnehmern aufgerufen, die auf Grundlage des bereitgestellten Tokens zur Veröffentlichung berechtigt sind.

Hier folgt ein Beispiel für eine Implementierung:

```
@Override
```

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

Sie ist für eine normale Video-Chat-Anwendung gedacht, bei der Benutzer immer etwas veröffentlichen möchten. Sie können die Audio- und Videowiedergabe stummschalten und die Stummschaltung aufheben, um umgehend ausgeblendet oder gesehen/gehört zu werden. (Sie können auch „Veröffentlichen/Veröffentlichung aufheben“ verwenden, was aber viel langsamer ist. „Stummschalten/Stummschalten aufheben“ ist für Anwendungsfälle vorzuziehen, in denen eine häufige Änderung der Sichtbarkeit wünschenswert ist.)

Auswählen von Streams zur Veröffentlichung

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

Beim Veröffentlichen wird hiermit bestimmt, welche Audio- und Videostreams veröffentlicht werden sollen. Dieser Punkt wird später unter Veröffentlichen eines Medienstreams ausführlicher behandelt.

Aktualisieren der Strategie

Die Strategie soll dynamisch sein: Die von einer der oben genannten Funktionen zurückgegebenen Werte lassen sich jederzeit ändern. Wenn die Hostanwendung beispielsweise erst veröffentlichen soll, wenn der Endbenutzer auf eine Schaltfläche tippt, können Sie eine Variable aus `shouldPublishFromParticipant` zurückgeben (zum Beispiel `hasUserTappedPublishButton`). Wenn sich diese Variable aufgrund einer Interaktion des Endbenutzers ändert, signalisieren Sie dem SDK per Aufruf von `stage.refreshStrategy()`, dass es die Strategie nach den neuesten Werten abfragen und nur Dinge anwenden soll, die sich geändert haben. Wenn das SDK feststellt, dass sich der Wert `shouldPublishFromParticipant` geändert hat, startet es den Veröffentlichungsprozess. Wenn alle Funktionen bei einer SDK-Abfrage den gleichen Wert zurückgeben wie zuvor, werden mit dem Aufruf von `refreshStrategy` keine Änderungen an der Bühne durchgeführt.

Ändert sich der Rückgabewert von `shouldSubscribeToParticipant` von `AUDIO_VIDEO` in `AUDIO_ONLY`, wird der Videostream für alle Teilnehmer mit geänderten Rückgabewerten entfernt, sofern zuvor ein Videostream vorhanden war.

Im Allgemeinen nutzt die Bühne die Strategie, um den Unterschied zwischen der vorherigen und der aktuellen Strategie am effizientesten anzuwenden. Dabei muss sich die Hostanwendung nicht um die ganzen Status kümmern, die für eine ordnungsgemäße Verwaltung erforderlich sind. Stellen Sie sich den Aufruf von `stage.refreshStrategy()` daher als einen ressourcenschonenden Vorgang vor, da nur bei einer Änderung der Strategie etwas unternommen wird.

Renderer

Die Schnittstelle `StageRenderer` teilt der Hostanwendung den Status der Bühne mit. Aktualisierungen in der Benutzeroberfläche der Hostanwendung können in der Regel vollständig über die vom Renderer bereitgestellten Ereignisse gesteuert werden. Der Renderer stellt die folgenden Funktionen bereit:

```
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);
```

Für die meisten dieser Methoden werden die entsprechende `Stage` und `ParticipantInfo` bereitgestellt.

Es wird nicht erwartet, dass sich die vom Renderer bereitgestellten Informationen auf die Rückgabewerte der Strategie auswirken. Es wird beispielsweise nicht erwartet, dass sich der Rückgabewert von `shouldSubscribeToParticipant` beim Aufruf von `onParticipantPublishStateChanged` ändert. Wenn die Hostanwendung einen bestimmten Teilnehmer abonnieren möchte, muss sie unabhängig von dessen Veröffentlichungsstatus den gewünschten Abonnementtyp zurückgeben. Das SDK muss dafür sorgen, dass entsprechend dem Status der Bühne und dem gewünschten Status der Strategie zum richtigen Zeitpunkt gehandelt wird.

Der `StageRenderer` kann der Bühnenklasse angefügt werden:

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

Hinweis: Nur veröffentlichende Teilnehmer lösen `onParticipantJoined` aus. Wenn Teilnehmer die Veröffentlichung beenden oder die Bühnensitzung verlassen, wird `onParticipantLeft` ausgelöst.

## Veröffentlichen eines Medienstreams

Lokale Geräte wie eingebaute Mikrofone und Kameras werden über `DeviceDiscovery` erkannt. Hier folgt ein Beispiel für die Auswahl der nach vorne gerichteten Kamera und des Standardmikrofons und deren anschließende Rückgabe als `LocalStageStreams` zur Veröffentlichung durch das 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;
}
```

## Anzeigen und Entfernen von Teilnehmern

Nach Abschluss von Abonnements erhalten Sie über die Funktion onStreamsAdded des Renderers eine Reihe von StageStream-Objekten. Sie können die Vorschau von einem ImageStageStream abrufen:

```
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);
```

Außerdem können Sie die Statistiken des Audiolevels von einem AudioStageStream abrufen:

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

Wenn ein Teilnehmer die Veröffentlichung beendet oder dessen Abonnement beendet wird, wird die Funktion onStreamsRemoved mit den Streams aufgerufen, die entfernt wurden. Hostanwendungen sollten dies als Signal nutzen, um den Videostream des Teilnehmers aus der Ansichtshierarchie zu entfernen.

`onStreamsRemoved` wird für alle Szenarien aufgerufen, in denen ein Stream entfernt werden könnte, darunter:

- Der Remote-Teilnehmer beendet die Veröffentlichung.
- Ein lokales Gerät beendet das Abonnement oder ändert das Abonnement von `AUDIO_VIDEO` in `AUDIO_ONLY`.
- Der Remote-Teilnehmer verlässt die Bühne.
- Der lokale Teilnehmer verlässt die Bühne.

Da `onStreamsRemoved` bei allen Szenarien aufgerufen wird, ist keine benutzerdefinierte Geschäftslogik erforderlich, um Teilnehmer beim remoten oder lokalen Verlassen aus der Benutzeroberfläche zu entfernen.

## Stummschalten von Medienstreams und Aufheben der Stummschaltung

`LocalStageStream`-Objekte verfügen über eine `setMuted`-Funktion, die das Stummschalten des Streams steuert. Diese Funktion kann für den Stream aufgerufen werden, bevor oder nachdem er von der Strategiefunktion `streamsToPublishForParticipant` zurückgegeben wird.

Wichtig: Wenn nach einem Aufruf von `refreshStrategy` eine neue `LocalStageStream`-Objekt-Instance von `streamsToPublishForParticipant` zurückgegeben wird, wird der Stummschaltungsstatus des neuen Streamobjekts auf die Bühne angewendet. Seien Sie vorsichtig beim Erstellen neuer `LocalStageStream`-Instances, um sicherzustellen, dass der erwartete Stummschaltungsstatus beibehalten wird.

## Überwachen des Medien-Stummschaltungsstatus von Remote-Teilnehmern

Wenn ein Teilnehmer den Stummschaltungsstatus seines Video- oder Audiostreams ändert, wird die Funktion `onStreamMutedChanged` des Renderers mit einer Liste der Streams aufgerufen, die sich geändert haben. Verwenden Sie die Methode `getMuted` für `StageStream`, um die Benutzeroberfläche entsprechend zu aktualisieren.

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

```
 }
}
```

## Abrufen von WebRTC-Statistiken

Um die neuesten WebRTC-Statistiken für einen veröffentlichten oder abonnierten Stream abzurufen, verwenden Sie `requestRTCStats` für `StageStream`. Nach Abschluss einer Erfassung erhalten Sie Statistiken über den `StageStream.Listener`, der für `StageStream` eingestellt werden kann.

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

## Abrufen von Teilnehmerattributen

Wenn Sie Attribute in der Endpunktanfrage `CreateParticipantToken` angeben, können Sie die Attribute in den Eigenschaften von `ParticipantInfo` einsehen:

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

## Fortsetzen der Sitzung im Hintergrund

Wenn die App in den Hintergrund wechselt, können Sie die Veröffentlichung beenden oder das Abonnement auf das Audio anderer Remote-Teilnehmer beschränken. Dazu aktualisieren Sie die Implementierung Ihrer `Strategy`, um die Veröffentlichung zu beenden und `AUDIO_ONLY` zu abonnieren (oder gegebenenfalls `NONE`).

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

## Aktivieren/Deaktivieren der mehrschichtigen Kodierung mit Simulcast

Bei der Veröffentlichung eines Medienstreams überträgt das SDK Videostreams in hoher und niedriger Qualität, sodass externe Teilnehmer den Stream abonnieren können, auch wenn sie nur über eine begrenzte Downlink-Bandbreite verfügen. Die mehrschichtige Kodierung mit Simulcast ist standardmäßig aktiviert. Sie können sie deaktivieren, indem Sie die `StageVideoConfiguration.Simulcast`-Klasse verwenden:

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

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

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

## Einschränkungen der Videokonfiguration

Das SDK unterstützt kein Erzwingen des Hoch- oder Querformats mit `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)`. Im Hochformat wird die kleinere Dimension als Breite verwendet, im Querformat als Höhe. Das bedeutet, dass die folgenden beiden Aufrufe von `setSize` die gleiche Auswirkung auf die Videokonfiguration haben:

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

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

## Umgang mit Netzwerkproblemen

Bei Unterbrechung der Netzwerkverbindung des lokalen Geräts versucht das SDK intern, die Verbindung ohne Benutzeraktion wiederherzustellen. In einigen Fällen ist das SDK nicht erfolgreich, weshalb eine Benutzeraktion erforderlich ist. Es gibt zwei Hauptfehler im Zusammenhang mit der Unterbrechung der Netzwerkverbindung:

- Fehlercode 1400, Meldung: „PeerConnection ist aufgrund eines unbekannten Netzwerkfehlers verloren“
- Fehlercode 1300, Meldung: „Die Zahl der Wiederholungsversuche ist ausgeschöpft.“

Wenn der erste Fehler empfangen wird, der zweite jedoch nicht, ist das SDK immer noch mit der Bühne verbunden und versucht, die Verbindungen automatisch wiederherzustellen. Zur Sicherheit können Sie `refreshStrategy` ohne Änderungen an den Rückgabewerten der Strategiemethode aufrufen, um einen manuellen Neuverbindungsversuch auszulösen.

Wenn der zweite Fehler empfangen wird, sind die Neuverbindungsversuche des SDK fehlgeschlagen und das lokale Gerät ist nicht mehr mit der Bühne verbunden. Versuchen Sie in diesem Fall, der Bühne erneut beizutreten, indem Sie `join` aufrufen, nachdem die Netzwerkverbindung wiederhergestellt wurde.

Im Allgemeinen deutet das Auftreten von Fehlern nach dem erfolgreichen Beitritt zu einer Bühne darauf hin, dass das SDK beim Wiederherstellen einer Verbindung nicht erfolgreich war.

Erstellen Sie ein neues `Stage`-Objekt und versuchen Sie, der Bühne beizutreten, wenn sich die Netzwerkbedingungen verbessern.

### Verwenden von Bluetooth-Mikrofonen

Um mit Bluetooth-Mikrofongeräten zu veröffentlichen, müssen Sie eine Bluetooth-SCO-Verbindung herstellen:

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

## Bekannte Probleme und Problemumgehungen

- Wenn ein Android-Gerät in den Ruhezustand wechselt und aufwacht, befindet sich die Vorschau möglicherweise in einem eingefrorenen Zustand.

  Problemumgehung: Erstellen und nutzen Sie eine neue `Stage`.
- Wenn ein Teilnehmer mit einem Token beitritt, das von einem anderen Teilnehmer verwendet wird, wird die erste Verbindung ohne einen bestimmten Fehler getrennt.

  Problemumgehung: Keine.
- Es gibt ein seltenes Problem, bei dem der Publisher etwas veröffentlicht, der Veröffentlichungsstatus, den Subscriber erhalten, jedoch `inactive` lautet.

  Problemumgehung: Versuchen Sie, die Sitzung zu verlassen und ihr wieder beizutreten. Wenn das Problem weiterhin besteht, erstellen Sie ein neues Token für den Publisher.
- Während einer Bühnensitzung kann zeitweise ein seltenes Problem mit Tonverzerrungen auftreten, in der Regel bei längeren Anrufen.

  Problemumgehung: Der Teilnehmer mit dem verzerrtem Ton kann die Sitzung entweder verlassen und erneut beitreten oder die Veröffentlichung des Audios aufheben und dann erneut veröffentlichen.
- Externe Mikrofone werden bei der Veröffentlichung auf einer Bühne nicht unterstützt.

  Problemumgehung: Verwenden Sie kein über USB angeschlossenes externes Mikrofon, um etwas auf einer Bühne zu veröffentlichen.

- Das Veröffentlichen auf einer Bühne mit der Bildschirmfreigabe über `createSystemCaptureSources` wird nicht unterstützt.

  Problemumgehung: Verwalten Sie die Systemerfassung manuell, indem Sie benutzerdefinierte Bild- und Audioeingangsquellen verwenden.

- Wenn eine `ImagePreviewView` in einem übergeordneten Element entfernt wird (`removeView()` wird z. B. im übergeordneten Element aufgerufen), wird die `ImagePreviewView` sofort freigegeben. Die `ImagePreviewView` zeigt keine Frames an, wenn sie einer anderen übergeordneten Ansicht hinzugefügt wird.

  Problemumgehung: Fordern Sie mit `getPreview` eine andere Vorschau an.

- Beim Beitritt zu einer Bühne mit einem Samsung Galaxy S22/+ mit Android 12 tritt möglicherweise ein 1401-Fehler auf. Das lokale Gerät kann der Bühne nicht beitreten oder tritt ihr bei, hat aber keinen Ton.

  Problemumgehung: Führen Sie ein Upgrade auf Android 13 durch.

- Beim Beitritt zu einer Bühne mit einem Nokia X20 unter Android 13 lässt sich die Kamera möglicherweise nicht öffnen und es wird eine Ausnahme ausgelöst.

  Problemumgehung: Keine.

- Geräte mit dem MediaTek Helio-Chipsatz rendern Videos von Remote-Teilnehmern möglicherweise nicht richtig.

  Problemumgehung: Keine.

- Auf einigen Geräten wählt das Betriebssystem möglicherweise ein anderes Mikrofon als das, das im SDK ausgewählt wurde. Das liegt daran, dass das Amazon IVS Broadcast SDK nicht steuern kann, wie die Audioroute `VOICE_COMMUNICATION` definiert wird, da sie je nach Gerätehersteller unterschiedlich ist.

  Problemumgehung: Keine.

- Einige Android-Video-Encoder können nicht mit einer Videogröße unter 176x176 konfiguriert werden. Die Konfiguration einer kleineren Größe verursacht einen Fehler und verhindert das Streaming.

  Problemumgehung: Konfigurieren Sie die Videogröße nicht so, dass sie kleiner als 176 x 176 ist.

# Fehlerbehandlung

## Schwerwiegende und nicht schwerwiegende Fehler

Das Fehlerobjekt hat das boolesche Feld „ist fatal“ von `BroadcastException`.

Im Allgemeinen hängen schwerwiegende Fehler mit der Verbindung zum Stages-Server zusammen (entweder kann eine Verbindung nicht hergestellt werden oder sie ist verloren gegangen und kann nicht wiederhergestellt werden). Die Anwendung sollte die Phase neu erstellen und erneut beitreten, ggf. mit einem neuen Token oder wenn die Konnektivität des Geräts wiederhergestellt ist.

Fehler, die nicht schwerwiegend sind, hängen in der Regel mit dem Status „Veröffentlichen/Abonnieren“ zusammen und werden vom SDK behandelt, das den Vorgang zum Veröffentlichen/Abonnieren erneut versucht.

Sie können diese Eigenschaft überprüfen:

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

## Beitrittsfehler

Fehlerhaft formatiertes Token

Dies passiert, wenn das Phasen-Token falsch formatiert ist.

Das SDK löst bei einem Aufruf von eine Java-Ausnahme mit dem Fehlercode = 1000 und fatal = true aus. `stage.join`

Aktion: Erstellen Sie ein gültiges Token und versuchen Sie erneut, Mitglied zu werden.

Abgelaufenes Token

Dies passiert, wenn das Phasen-Token abgelaufen ist.

Das SDK löst bei einem Aufruf von eine Java-Ausnahme mit dem Fehlercode = 1001 und fatal = true aus. `stage.join`

Aktion: Erstellen Sie ein neues Token und versuchen Sie erneut, Mitglied zu werden.

Ungültiges oder widerrufenes Token

Dies passiert, wenn das Phasen-Token nicht falsch formatiert ist, sondern vom Stages-Server zurückgewiesen wird. Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK ruft `onConnectionStateChanged` mit einer Ausnahme auf, mit dem Fehlercode = 1026 und fatal = true.

Aktion: Erstellen Sie ein gültiges Token und versuchen Sie erneut beizutreten.

Netzwerkfehler beim ersten Beitritt

Dies passiert, wenn das SDK den Stages-Server nicht kontaktieren kann, um eine Verbindung herzustellen. Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK ruft `onConnectionStateChanged` mit einer Ausnahme auf, mit dem Fehlercode = 1300 und fatal = true.

Handlung: Warten Sie, bis die Konnektivität des Geräts wiederhergestellt ist, und versuchen Sie erneut, eine Verbindung herzustellen.

Netzwerkfehler, wenn bereits eine Verbindung hergestellt wurde

Wenn die Netzwerkverbindung des Geräts ausfällt, verliert das SDK möglicherweise die Verbindung zu den Phasen-Servern. Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK ruft `onConnectionStateChanged` mit einer Ausnahme auf, mit dem Fehlercode = 1300 und fatal = true.

Handlung: Warten Sie, bis die Konnektivität des Geräts wiederhergestellt ist, und versuchen Sie erneut, eine Verbindung herzustellen.

## Fehler beim Veröffentlichen/Abonnieren

Anfänglich

Es gibt mehrere Arten von Fehlern:

- MultihostSessionOfferCreationFailPublish (1020)
- MultihostSessionOfferCreationFailSubscribe (1021)

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

Diese werden asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK wiederholt den Vorgang für eine begrenzte Anzahl von Malen. Bei Wiederholungen ist der Status „Veröffentlichen/Abonnieren“ `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Wenn die Wiederholungsversuche erfolgreich sind, ändert sich der Status auf `PUBLISHED` / `SUBSCRIBED`.

Das SDK ruft `onError` mit dem entsprechenden Fehlercode und fatal = false auf.

Aktion: Es ist keine Aktion erforderlich, da das SDK es automatisch wiederholt. Optional kann die Anwendung die Strategie aktualisieren, um weitere Wiederholungsversuche zu erzwingen.

Bereits eingerichtet, dann gescheitert

Eine Veröffentlichung oder ein Abonnement kann nach der Einrichtung fehlschlagen, was höchstwahrscheinlich auf einen Netzwerkfehler zurückzuführen ist. Fehlercode 1400, Meldung: „Die Peer-Verbindung wurde aufgrund eines unbekannten Netzwerkfehlers unterbrochen.“

Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK versucht erneut, den Vorgang zu veröffentlichen/abonnieren. Bei Wiederholungen ist der Status „Veröffentlichen/Abonnieren“ `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Wenn die Wiederholungsversuche erfolgreich sind, ändert sich der Status auf `PUBLISHED` / `SUBSCRIBED`.

Das SDK ruft `onError` mit dem Fehlercode = 1400 und fatal = false auf.

Aktion: Es ist keine Aktion erforderlich, da das SDK es automatisch wiederholt. Optional kann die Anwendung die Strategie aktualisieren, um weitere Wiederholungsversuche zu erzwingen. Im Falle eines vollständigen Verbindungsverlusts ist es wahrscheinlich, dass auch die Verbindung zu Stages fehlschlägt.

# IVS-Broadcast-SDK: iOS-Handbuch (Echtzeit-Streaming)

Das iOS-Broadcast-SDK für IVS Echtzeit-Streaming ermöglicht es den Teilnehmern, Videos auf iOS zu senden und zu empfangen.

Das Modul `AmazonIVSBroadcast` implementiert die in diesem Dokument beschriebene Schnittstelle. Folgende Operationen werden unterstützt:

- Einer Bühne beitreten
- Medien für andere Teilnehmer auf der Bühne veröffentlichen
- Medien anderer Teilnehmer auf der Bühne abonnieren
- Auf der Bühne veröffentlichte Videos und Audios verwalten und überwachen
- WebRTC-Statistiken für jede Peer-Verbindung beziehen
- Alle Vorgänge aus dem IVS-Streaming-SDK für iOS-Übertragungen mit niedriger Latenz

Neueste Version des iOS-Broadcast-SDK: 1.14.1 (Versionshinweise)

Referenzdokumentation: Informationen zu den wichtigsten Methoden, die im Amazon IVS iOS Broadcast SDK verfügbar sind, finden Sie in der Referenzdokumentation unter https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/ iOS/.

Beispielcode: Siehe das iOS-Beispiel-Repository auf GitHub: https://github.com/aws-samples/amazon-ivs-broadcast-ios-sample .

Plattform-Anforderungen: iOS 14 oder höher.

# Erste Schritte

## Installieren Sie die Bibliothek

Wir empfehlen Ihnen, das Broadcast-SDK über zu integrieren CocoaPods. (Alternativ können Sie die Framework manuell zu Ihrem Projekt hinzufügen.)

Empfohlen: Integrieren des Broadcast-SDK (CocoaPods)

Die Echtzeitfunktionalität wird als Unterspezifikation des iOS-Streaming-Broadcast-SDK mit niedriger Latenz veröffentlicht. So können Kunden je nach ihren Feature-Ánforderungen wählen, ob sie sie einbeziehen oder ausschließen möchten. Wenn die Funktionalität einbezogen wird, erhöht sich die Paketgröße.

Releases werden über CocoaPods unter dem Namen veröffentlicht`AmazonIVSBroadcast`. Fügen Sie diese Abhängigkeit zu Ihrem Podfile hinzu:

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

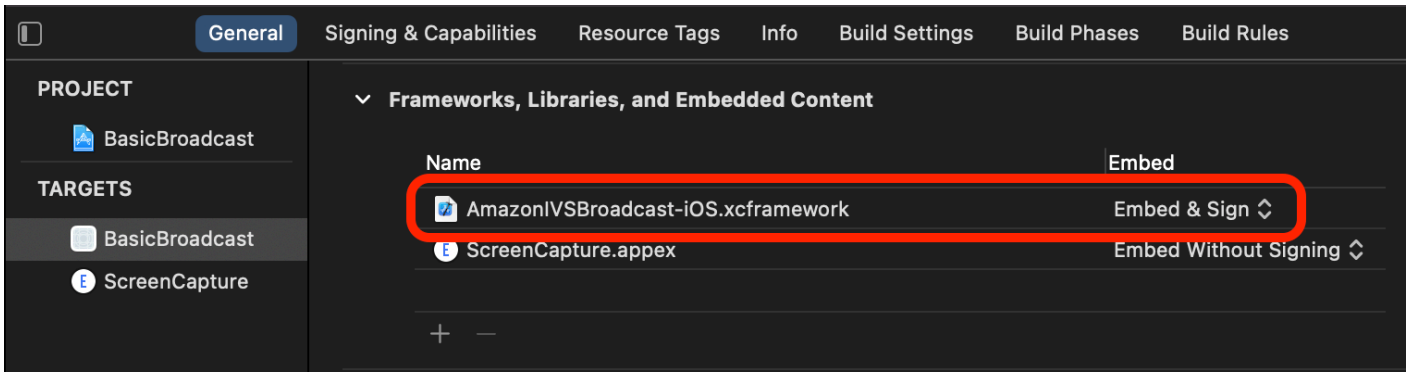
Führen Sie `pod install` aus und das SDK wird in Ihrem `.xcworkspace` verfügbar sein.

Wichtig: Das IVS-Echtzeit-Streaming-Broadcast-SDK (d. h. mit der Stage-Unterspezifikation) beinhaltet alle Features des IVS-Streaming-Broadcast-SDK mit niedriger Latenz. Es ist nicht möglich, beide SDKs in dasselbe Projekt zu integrieren. Wenn Sie die Stufenunterspezifikation über CocoaPods zu Ihrem Projekt hinzufügen, stellen Sie sicher, dass Sie alle anderen Zeilen in der Podfile entfernen, die enthält`AmazonIVSBroadcast`. Zum Beispiel darf Ihre Podfile nicht beide der folgenden Zeilen enthalten:

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

Manuelles Installieren der Framework

1. Laden Sie die neueste Version von https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip herunter.
2. Extrahieren Sie den Inhalt des Archivs. `AmazonIVSBroadcast.xcframework` enthält das SDK für Gerät und Simulator.
3. Betten Sie `AmazonIVSBroadcast.xcframework` ein, indem Sie es in den Abschnitt Frameworks, Bibliotheken und eingebettete Inhalte auf der Registerkarte Allgemein für Ihr Anwendungsziel ziehen.




## Berechtigungen anfordern

Ihre App muss die Berechtigung für den Zugriff auf die Kamera und das Mikrofon des Benutzers anfordern. (Dies ist nicht spezifisch für Amazon IVS; es ist für jede Anwendung erforderlich, die Zugriff auf Kameras und Mikrofone benötigt.)

Hier prüfen wir, ob der Benutzer bereits Berechtigungen erteilt hat und wenn nicht, fragen wir nach ihnen:

```
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.
}
```

Sie müssen dies sowohl für `.video`- als auch für `.audio`-Medientypen tun, wenn Sie auf Kameras bzw. Mikrofone zugreifen möchten.

Sie müssen außerdem Einträge für `NSCameraUsageDescription` und `NSMicrophoneUsageDescription` zu Ihrem `Info.plist` hinzufügen. Andernfalls stürzt Ihre App ab, wenn Sie versuchen, Berechtigungen anzufordern.

### Deaktivieren des Idle-Timers der Anwendung

Dies ist zwar optional, wird aber empfohlen. Es verhindert, dass Ihr Gerät in den Ruhezustand versetzt, während Sie das Broadcast-SDK verwenden, was die Übertragung unterbrechen würde.

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

## Publishing und Subscribing

### Konzepte

Drei Kernkonzepte liegen der Echtzeit-Funktionalität zugrunde: Stage, Strategie und Renderer. Das Designziel besteht in der Minimierung der Menge an clientseitiger Logik, die für die Entwicklung eines funktionierenden Produkts erforderlich ist.

Stufe

Die Klasse `IVSStage` ist der Hauptinteraktionspunkt zwischen der Hostanwendung und dem SDK. Die Klasse stellt die Bühne selbst dar und dient dazu, der Bühne beizutreten und sie zu verlassen. Für das Erstellen einer Bühne oder das Beitreten ist eine gültige, noch nicht abgelaufene Token-Zeichenfolge aus der Steuerebene erforderlich (dargestellt als `token`). Einer Bühne beizutreten und sie zu verlassen, ist ganz einfach.

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

try stage.join()

stage.leave()
```

In der Klasse `IVSStage` erfolgt auch das Anhängen des `IVSStageRenderer` und `IVSErrorDelegate`:

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

Strategie

Über das Protokoll `IVSStageStrategy` kann die Hostanwendung dem SDK den gewünschten Status der Bühne mitteilen. Drei Funktionen müssen implementiert werden: `shouldSubscribeToParticipant`, `shouldPublishParticipant` und `streamsToPublishForParticipant`. Alle werden im Folgenden behandelt.

Abonnieren von Teilnehmern

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

Wenn ein Remote-Teilnehmer einer Bühne beitritt, fragt das SDK die Hostanwendung nach dessen gewünschtem Abonnementstatus. Die Optionen lauten `.none`, `.audioOnly` und `.audioVideo`. Wenn ein Wert für diese Funktion zurückgegeben wird, muss sich die Hostanwendung nicht um den Veröffentlichungs-, den aktuellen Abonnement- oder den Verbindungsstatus des Bühne kümmern. Bei Rückgabe von `.audioVideo` wartet das SDK mit dem Abonnieren, bis der Remote-Teilnehmer etwas veröffentlicht. Außerdem aktualisiert das SDK die Hostanwendung während des gesamten Prozesses über den Renderer.

Hier folgt ein Beispiel für eine Implementierung:

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

Hierbei handelt es sich um die vollständige Implementierung dieser Funktion für eine Hostanwendung, bei der sich alle Teilnehmer stets gegenseitig sehen sollen; z. B. eine Video-Chat-Anwendung.

Weitergehende Implementierungen sind ebenfalls möglich. Nutzen Sie die Eigenschaft `attributes` für `IVSParticipantInfo`, um Teilnehmer anhand der vom Server bereitgestellten Attribute selektiv zu abonnieren:

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

Hiermit kann eine Bühne erstellt werden, auf der Moderatoren alle Gäste überwachen können, ohne selbst gesehen oder gehört zu werden. Die Hostanwendung könnte eine zusätzliche Geschäftslogik nutzen, damit Moderatoren sich gegenseitig sehen können, für Gäste aber unsichtbar bleiben.

Veröffentlichen

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

Sobald die Verbindung zur Bühne hergestellt ist, überprüft das SDK per Anfrage an die Hostanwendung, ob ein bestimmter Teilnehmer etwas veröffentlichen soll. Dies wird nur bei lokalen Teilnehmern aufgerufen, die auf Grundlage des bereitgestellten Tokens zur Veröffentlichung berechtigt sind.

Hier folgt ein Beispiel für eine Implementierung:

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

Sie ist für eine normale Video-Chat-Anwendung gedacht, bei der Benutzer immer etwas veröffentlichen möchten. Sie können die Audio- und Videowiedergabe stummschalten und die Stummschaltung aufheben, um umgehend ausgeblendet oder gesehen/gehört zu werden. (Sie können auch „Veröffentlichen/Veröffentlichung aufheben“ verwenden, was aber viel langsamer ist. „Stummschalten/Stummschalten aufheben“ ist für Anwendungsfälle vorzuziehen, in denen eine häufige Änderung der Sichtbarkeit wünschenswert ist.)

Auswählen von Streams zur Veröffentlichung

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

Beim Veröffentlichen wird hiermit bestimmt, welche Audio- und Videostreams veröffentlicht werden sollen. Dieser Punkt wird später unter Veröffentlichen eines Medienstreams ausführlicher behandelt.

Aktualisieren der Strategie

Die Strategie soll dynamisch sein: Die von einer der oben genannten Funktionen zurückgegebenen Werte lassen sich jederzeit ändern. Wenn die Hostanwendung beispielsweise erst veröffentlichen soll, wenn der Endbenutzer auf eine Schaltfläche tippt, können Sie eine Variable aus `shouldPublishParticipant` zurückgeben (zum Beispiel `hasUserTappedPublishButton`). Wenn sich diese Variable aufgrund einer Interaktion des Endbenutzers ändert, signalisieren Sie dem SDK per Aufruf von `stage.refreshStrategy()`, dass es die Strategie nach den neuesten Werten abfragen und nur Dinge anwenden soll, die sich geändert haben. Wenn das SDK feststellt, dass sich der Wert `shouldPublishParticipant` geändert hat, startet es den Veröffentlichungsprozess. Wenn alle Funktionen bei einer SDK-Abfrage den gleichen Wert zurückgeben wie zuvor, werden mit dem Aufruf von `refreshStrategy` keine Änderungen an der Bühne durchgeführt.

Ändert sich der Rückgabewert von `shouldSubscribeToParticipant` von `.audioVideo` in `.audioOnly`, wird der Videostream für alle Teilnehmer mit geänderten Rückgabewerten entfernt, sofern zuvor ein Videostream vorhanden war.

Im Allgemeinen nutzt die Bühne die Strategie, um den Unterschied zwischen der vorherigen und der aktuellen Strategie am effizientesten anzuwenden. Dabei muss sich die Hostanwendung nicht um die

ganzen Status kümmern, die für eine ordnungsgemäße Verwaltung erforderlich sind. Stellen Sie sich den Aufruf von `stage.refreshStrategy()` daher als einen ressourcenschonenden Vorgang vor, da nur bei einer Änderung der Strategie etwas unternommen wird.

Renderer

Das Protokoll `IVSStageRenderer` teilt der Hostanwendung den Status der Bühne mit. Aktualisierungen in der Benutzeroberfläche der Hostanwendung können in der Regel vollständig über die vom Renderer bereitgestellten Ereignisse gesteuert werden. Der Renderer stellt die folgenden Funktionen bereit:

```
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?)
```

Es wird nicht erwartet, dass sich die vom Renderer bereitgestellten Informationen auf die Rückgabewerte der Strategie auswirken. Es wird beispielsweise nicht erwartet, dass sich der Rückgabewert von `shouldSubscribeToParticipant` beim Aufruf von `participant:didChangePublishState` ändert. Wenn die Hostanwendung einen bestimmten Teilnehmer abonnieren möchte, muss sie unabhängig von dessen Veröffentlichungsstatus den gewünschten Abonnementtyp zurückgeben. Das SDK muss dafür sorgen, dass entsprechend dem Status der Bühne und dem gewünschten Status der Strategie zum richtigen Zeitpunkt gehandelt wird.

Hinweis: Nur veröffentlichende Teilnehmer lösen `participantDidJoin` aus. Wenn Teilnehmer die Veröffentlichung beenden oder die Bühnensitzung verlassen, wird `participantDidLeave` ausgelöst.

## Veröffentlichen eines Medienstreams

Lokale Geräte wie eingebaute Mikrofone und Kameras werden über `IVSDeviceDiscovery` erkannt. Hier folgt ein Beispiel für die Auswahl der nach vorne gerichteten Kamera und des Standardmikrofons und deren anschließende Rückgabe als `IVSLocalStageStreams` zur Veröffentlichung durch das 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]
}
```

## Anzeigen und Entfernen von Teilnehmern

Nach Abschluss von Abonnements erhalten Sie über die Funktion `didAddStreams` des Renderers eine Reihe von `IVSStageStream`-Objekten. Um Statistiken des Audiolevels zu diesem Teilnehmer in der Vorschau anzuzeigen oder abzurufen, können Sie über den Stream auf das zugrunde liegende `IVSDevice`-Objekt zugreifen:

```
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 */
    })
}
```

Wenn ein Teilnehmer die Veröffentlichung beendet oder dessen Abonnement beendet wird, wird die Funktion `didRemoveStreams` mit den Streams aufgerufen, die entfernt wurden. Hostanwendungen sollten dies als Signal nutzen, um den Videostream des Teilnehmers aus der Ansichtshierarchie zu entfernen.

`didRemoveStreams` wird für alle Szenarien aufgerufen, in denen ein Stream entfernt werden könnte, darunter:

- Der Remote-Teilnehmer beendet die Veröffentlichung.
- Ein lokales Gerät beendet das Abonnement oder ändert das Abonnement von `.audioVideo` in `.audioOnly`.
- Der Remote-Teilnehmer verlässt die Bühne.
- Der lokale Teilnehmer verlässt die Bühne.

Da `didRemoveStreams` bei allen Szenarien aufgerufen wird, ist keine benutzerdefinierte Geschäftslogik erforderlich, um Teilnehmer beim remoten oder lokalen Verlassen aus der Benutzeroberfläche zu entfernen.

### Stummschalten von Medienstreams und Aufheben der Stummschaltung

`IVSLocalStageStream`-Objekte verfügen über eine `setMuted`-Funktion, die das Stummschalten des Streams steuert. Diese Funktion kann für den Stream aufgerufen werden, bevor oder nachdem er von der Strategiefunktion `streamsToPublishForParticipant` zurückgegeben wird.

Wichtig: Wenn nach einem Aufruf von `refreshStrategy` eine neue `IVSLocalStageStream`-Objekt-Instance von `streamsToPublishForParticipant` zurückgegeben wird, wird der Stummschaltungsstatus des neuen Streamobjekts auf die Bühne angewendet. Seien Sie vorsichtig beim Erstellen neuer `IVSLocalStageStream`-Instances, um sicherzustellen, dass der erwartete Stummschaltungsstatus beibehalten wird.

### Überwachen des Medien-Stummschaltungsstatus von Remote-Teilnehmern

Wenn ein Teilnehmer den Stummschaltungsstatus seines Video- oder Audiostreams ändert, wird die Funktion `didChangeMutedStreams` des Renderers mit einer Gruppe der Streams aufgerufen,

die sich geändert haben. Verwenden Sie die Eigenschaft `isMuted` für `IVSStageStream`, um die Benutzeroberfläche entsprechend zu aktualisieren:

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

## Erstellen einer Bühnenkonfiguration

Die Werte der Videokonfiguration einer Bühne passen Sie mit `IVSLocalStageStreamVideoConfiguration` an:

```
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
```

## Abrufen von WebRTC-Statistiken

Um die neuesten WebRTC-Statistiken für einen veröffentlichten oder abonnierten Stream abzurufen, verwenden Sie `requestRTCStats` für `IVSStageStream`. Nach Abschluss einer Erfassung erhalten Sie Statistiken über den `IVSStageStreamDelegate`, der für `IVSStageStream` eingestellt werden kann. Um WebRTC-Statistiken kontinuierlich zu erfassen, rufen Sie diese Funktion für einen `Timer` auf.

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

### Abrufen von Teilnehmerattributen

Wenn Sie Attribute in der Endpunktanfrage `CreateParticipantToken` angeben, können Sie die Attribute in den Eigenschaften von `IVSParticipantInfo` einsehen:

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

### Fortsetzen der Sitzung im Hintergrund

Wenn die App in den Hintergrund wechselt, können Sie weiterhin auf der Bühne präsent sein, während Sie Remote-Ton hören. Es ist jedoch nicht möglich, das eigene Bild und den eigenen Ton weiterhin zu senden. Sie müssen die Implementierung Ihrer `IVSStrategy` aktualisieren, um die Veröffentlichung zu beenden und `.audioOnly` zu abonnieren (oder gegebenenfalls `.none`):

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

Anschließend rufen Sie `stage.refreshStrategy()` auf.

### Aktivieren/Deaktivieren der mehrschichtigen Kodierung mit Simulcast

Bei der Veröffentlichung eines Medienstreams überträgt das SDK Videostreams in hoher und niedriger Qualität, sodass externe Teilnehmer den Stream abonnieren können, auch wenn sie nur über eine begrenzte Downlink-Bandbreite verfügen. Die mehrschichtige Kodierung mit Simulcast ist standardmäßig aktiviert. Sie können es deaktivieren mit `IVSSimulcastConfiguration`:

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

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

// Other Stage implementation code
```

### Übertragung der Stage auf einen IVS-Kanal

Zum Übertragen einer Bühne erstellen Sie eine separate `IVSBroadcastSession` und folgen Sie dann den oben beschriebenen üblichen Anweisungen für die Übertragung mit dem SDK. Die Eigenschaft `device` für `IVSStageStream` ist entweder ein `IVSImageDevice` oder ein `IVSAudioDevice`, wie im obigen Snippet veranschaulicht. Diese können mit dem `IVSBroadcastSession.mixer` verbunden werden, um die gesamte Bühne in einem individuell anpassbaren Layout zu übertragen.

Optional können Sie eine Stage zusammenstellen und sie auf einen IVS-Kanal mit niedriger Latenz übertragen, um ein größeres Publikum zu erreichen. Sehen Sie Aktivierung mehrerer Hosts in einem Amazon-IVS-Stream im Benutzerhandbuch für IVS-Streaming mit niedriger Latenz.

## So wählt iOS Kameraauflösung und Bildrate

Die vom Broadcast-SDK verwaltete Kamera optimiert ihre Auflösung und Bildrate (frames-per-second, oder FPS), um die Temperaturentwicklung und den Verbrauch zu minimieren. In diesem Abschnitt wird erläutert, wie Auflösung und Bildrate ausgewählt werden, um Hostanwendungen bei der Optimierung für ihre Anwendungsfälle zu unterstützen.

Wenn ein `IVSLocalStageStream` mit einer `IVSCamera` erstellt wird, ist die Kamera für eine Bildrate von `IVSLocalStageStreamVideoConfiguration.targetFramerate` und eine Auflösung von `IVSLocalStageStreamVideoConfiguration.size` optimiert. Das Aufrufen von `IVSLocalStageStream.setConfiguration` aktualisiert die Kamera mit neueren Werten.

### Kameravorschau

Wenn Sie eine Vorschau einer `IVSCamera` erstellen, ohne sie mit einer `IVSBroadcastSession` oder `IVSStage` zu verbinden, wird standardmäßig eine Auflösung von 1080p und eine Bildrate von 60 Bildern pro Sekunde verwendet.

### Übertragen einer Stage

Bei der Verwendung einer `IVSBroadcastSession` zu Übertragung einer `IVSStage` versucht das SDK, die Kamera mit einer Auflösung und Bildrate zu optimieren, die die Kriterien beider Sitzungen erfüllen.

Wenn die Broadcast-Konfiguration beispielsweise auf eine Bildrate von 15 FPS und eine Auflösung von 1080p eingestellt ist, während die Stage eine Bildrate von 30 FPS und eine Auflösung von 720p hat, wählt das SDK eine Kamerakonfiguration mit einer Bildrate von 30 FPS und einer Auflösung von 1080p aus. Bei der `IVSBroadcastSession` fehlt jedes zweite Bild von der Kamera und die `IVSStage` skaliert das 1080p-Bild auf 720p herunter.

Wenn eine Host-Anwendung plant, sowohl eine `IVSBroadcastSession` als auch eine `IVSStage` zusammen mit einer Kamera zu verwenden, empfehlen wir, dass die Eigenschaften `targetFramerate` und `size` der jeweiligen Konfigurationen übereinstimmen. Eine Nichtübereinstimmung kann dazu führen, dass sich die Kamera während der Videoaufnahme selbst neu konfiguriert, was zu einer kurzen Verzögerung bei der Übertragung von Videos führt.

Wenn identische Werte nicht dem Anwendungsfall der Hostanwendung entsprechen, verhindert das Erstellen der Kamera mit höherer Qualität, dass sich die Kamera selbst neu konfiguriert, wenn die Sitzung mit niedrigerer Qualität hinzugefügt wird. Wenn Sie zum Beispiel mit 1080p und 30 Bildern pro Sekunde übertragen und später einer Stage beitreten, die auf 720p und 30 FPS eingestellt ist, konfiguriert sich die Kamera nicht selbst neu und die Videowiedergabe läuft ohne Unterbrechung weiter. Dies liegt daran, dass 720p kleiner oder gleich 1080p und 30 FPS kleiner oder gleich 30 FPS sind.

### Beliebige Bildraten, Auflösungen und Seitenverhältnisse

Die meisten Kameras können gängige Formate wie 720p bei 30 FPS oder 1080p bei 60 FPS exakt wiedergeben. Es ist jedoch unmöglich, alle Formate exakt wiederzugeben. Das Broadcast-SDK wählt die Kamerakonfiguration auf der Grundlage der folgenden Regeln aus (in der Reihenfolge ihrer Priorität):

1. Breite und Höhe der Auflösung sind größer oder gleich der gewünschten Auflösung, aber innerhalb dieser Beschränkung sind Breite und Höhe so klein wie möglich.
2. Die Bildrate ist größer oder gleich der gewünschten Bildrate, aber innerhalb dieser Beschränkung ist die Bildrate so niedrig wie möglich.
3. Das Seitenverhältnis entspricht dem gewünschten Seitenverhältnis.
4. Wenn es mehrere passende Formate gibt, wird das Format mit dem größten Sichtfeld verwendet.

Nachfolgend finden Sie zwei Beispiele:

- Die Host-Anwendung versucht, in 4K mit 120 FPS zu übertragen. Die ausgewählte Kamera unterstützt nur 4K bei 60 FPS oder 1080p bei 120 FPS. Das gewählte Format ist 4K bei 60 FPS, da die Auflösungsregel eine höhere Priorität hat als die Bildratenregel.
- Eine unregelmäßige Auflösung wird angefordert, 1910x1070. Die Kamera wird 1920x1080 verwenden. Seien Sie vorsichtig: Wenn Sie eine Auflösung wie 1921x1080 wählen, skaliert die Kamera auf die nächste verfügbare Auflösung (z. B. 2592x1944) hoch, was sich nachteilig auf die CPU- und Speicherbandbreite auswirkt.

### Was ist mit Android?

Android passt seine Auflösung oder Bildrate nicht wie iOS im laufenden Betrieb an, sodass dies keine Auswirkungen auf das Android-Broadcast-SDK hat.

## Bekannte Probleme und Problemumgehungen

- Das Ändern von Bluetooth-Audiorouten kann unvorhersehbar sein. Wenn Sie ein neues Gerät in der Mitte der Sitzung verbinden, kann iOS die Eingaberoute automatisch ändern oder nicht. Es ist auch nicht möglich, zwischen mehreren Bluetooth-Headsets zu wählen, die gleichzeitig verbunden sind. Dies geschieht sowohl bei normalen Broadcast- als auch bei Bühnensitzungen.

  Problemumgehung: Wenn Sie ein Bluetooth-Headset verwenden möchten, verbinden Sie es, bevor Sie den Broadcast oder die Bühne starten und lassen Sie es während der gesamten Sitzung verbunden.
- Teilnehmer, die ein iPhone 14, iPhone 14 Plus, iPhone 14 Pro oder iPhone 14 Pro Max nutzen, können bei anderen Teilnehmern ein Echo verursachen.

  Problemumgehung: Teilnehmer, die die betroffenen Geräte nutzen, können Kopfhörer verwenden, um das Echo bei anderen Teilnehmern zu vermeiden.
- Wenn ein Teilnehmer mit einem Token beitritt, das von einem anderen Teilnehmer verwendet wird, wird die erste Verbindung ohne einen bestimmten Fehler getrennt.

  Problemumgehung: Keine.
- Es gibt ein seltenes Problem, bei dem der Publisher etwas veröffentlicht, der Veröffentlichungsstatus, den Subscriber erhalten, jedoch `inactive` lautet.

  Problemumgehung: Versuchen Sie, die Sitzung zu verlassen und ihr wieder beizutreten. Wenn das Problem weiterhin besteht, erstellen Sie ein neues Token für den Publisher.

- Wenn ein Teilnehmer etwas veröffentlicht oder abonniert, kann selbst bei einem stabilen Netzwerk eine Fehlermeldung mit dem Code 1400 angezeigt werden. Sie weist darauf hin, dass die Verbindung aufgrund eines Netzwerkproblems unterbrochen wurde.

  Problemumgehung: Versuchen Sie, erneut zu veröffentlichen bzw. erneut zu abonnieren.
- Während einer Bühnensitzung kann zeitweise ein seltenes Problem mit Tonverzerrungen auftreten, in der Regel bei längeren Anrufen.

  Problemumgehung: Der Teilnehmer mit dem verzerrtem Ton kann die Sitzung entweder verlassen und erneut beitreten oder die Veröffentlichung des Audios aufheben und dann erneut veröffentlichen.

# Fehlerbehandlung

## Schwerwiegende und nicht schwerwiegende Fehler

Das Fehlerobjekt hat den booleschen Wert „ist fatal“. Dies ist ein Wörterbucheintrag unter `IVSBroadcastErrorIsFatalKey`, der einen booleschen Wert enthält.

Im Allgemeinen hängen schwerwiegende Fehler mit der Verbindung zum Stages-Server zusammen (entweder kann eine Verbindung nicht hergestellt werden oder sie ist verloren gegangen und kann nicht wiederhergestellt werden). Die Anwendung sollte die Phase neu erstellen und erneut beitreten, ggf. mit einem neuen Token oder wenn die Konnektivität des Geräts wiederhergestellt ist.

Fehler, die nicht schwerwiegend sind, hängen in der Regel mit dem Status „Veröffentlichen/Abonnieren“ zusammen und werden vom SDK behandelt, das den Vorgang zum Veröffentlichen/Abonnieren erneut versucht.

Sie können diese Eigenschaft überprüfen:

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

## Beitrittsfehler

Fehlerhaft formatiertes Token

Dies passiert, wenn das Phasen-Token falsch formatiert ist.

Das SDK löst eine Swift-Ausnahme mit dem Fehlercode = 1000 und IVS BroadcastErrorIsFatalKey = YES aus.

Aktion: Erstellen Sie ein gültiges Token und versuchen Sie erneut beizutreten.

Abgelaufenes Token

Dies passiert, wenn das Phasen-Token abgelaufen ist.

Das SDK löst eine Swift-Ausnahme mit dem Fehlercode = 1001 und IVS BroadcastErrorIsFatalKey = YES aus.

Aktion: Erstellen Sie ein neues Token und versuchen Sie erneut beizutreten.

Ungültiges oder widerrufenes Token

Dies passiert, wenn das Phasen-Token nicht falsch formatiert ist, sondern vom Stages-Server zurückgewiesen wird. Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK ruft `stage(didChange connectionState, withError error)` mit dem Fehlercode = 1026 und IVS BroadcastErrorIsFatalKey = YES auf.

Aktion: Erstellen Sie ein gültiges Token und versuchen Sie erneut beizutreten.

Netzwerkfehler beim ersten Beitritt

Dies passiert, wenn das SDK den Stages-Server nicht kontaktieren kann, um eine Verbindung herzustellen. Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK ruft `stage(didChange connectionState, withError error)` mit dem Fehlercode = 1300 und IVS BroadcastErrorIsFatalKey = JA auf.

Handlung: Warten Sie, bis die Konnektivität des Geräts wiederhergestellt ist, und versuchen Sie erneut, eine Verbindung herzustellen.

Netzwerkfehler, wenn bereits eine Verbindung hergestellt wurde

Wenn die Netzwerkverbindung des Geräts ausfällt, verliert das SDK möglicherweise die Verbindung zu den Phasen-Servern. Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK ruft `stage(didChange connectionState, withError error)` mit dem Fehlercode = 1300 und dem IVS-BroadcastErrorIsFatalKey Wert = JA auf.

Handlung: Warten Sie, bis die Konnektivität des Geräts wiederhergestellt ist, und versuchen Sie erneut, eine Verbindung herzustellen.

## Fehler beim Veröffentlichen/Abonnieren

Anfänglich

Es gibt mehrere Arten von Fehlern:

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

Diese werden asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK wiederholt den Vorgang für eine begrenzte Anzahl von Malen. Bei Wiederholungen ist der Status „Veröffentlichen/Abonnieren“ `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Wenn die Wiederholungsversuche erfolgreich sind, ändert sich der Status auf `PUBLISHED` / `SUBSCRIBED`.

Das SDK ruft `IVSErrorSourceDelegate:didEmitError` mit dem entsprechenden Fehlercode auf und IVS BroadcastErrorIsFatalKey = NO.

Aktion: Es ist keine Aktion erforderlich, da das SDK es automatisch wiederholt. Optional kann die Anwendung die Strategie aktualisieren, um weitere Wiederholungsversuche zu erzwingen.

Bereits eingerichtet, dann gescheitert

Eine Veröffentlichung oder ein Abonnement kann nach der Einrichtung fehlschlagen, was höchstwahrscheinlich auf einen Netzwerkfehler zurückzuführen ist. Fehlercode 1400, Meldung: „Die Peer-Verbindung wurde aufgrund eines unbekannten Netzwerkfehlers unterbrochen.“

Dies wird asynchron über den von der Anwendung bereitgestellten Phasen-Renderer gemeldet.

Das SDK versucht erneut, den Vorgang zu veröffentlichen/abonnieren. Bei Wiederholungen ist der Status „Veröffentlichen/Abonnieren“ `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`. Wenn die Wiederholungsversuche erfolgreich sind, ändert sich der Status auf `PUBLISHED` / `SUBSCRIBED`.

Das SDK ruft `didEmitError` mit dem Fehlercode = 1400 und IVS BroadcastErrorIsFatalKey = NO auf.

Aktion: Es ist keine Aktion erforderlich, da das SDK es automatisch wiederholt. Optional kann die Anwendung die Strategie aktualisieren, um weitere Wiederholungsversuche zu erzwingen. Im Falle eines vollständigen Verbindungsverlusts ist es wahrscheinlich, dass auch die Verbindung zu Stages fehlschlägt.

# IVS-Broadcast-SDK: Benutzerdefinierte Bildquellen (Echtzeit-Streaming)

Benutzerdefinierte Bildeingabequellen ermöglichen es einer Anwendung, eine eigene Bildeingabe für das Broadcast-SDK bereitzustellen, anstatt sich auf die voreingestellten Kameras oder die Bildschirmfreigabe zu beschränken. Eine benutzerdefinierte Bildquelle kann so einfach sein wie ein halbtransparentes Wasserzeichen oder eine statische „Bin gleich zurück“-Szene, oder es kann der Anwendung ermöglichen, zusätzliche benutzerdefinierte Verarbeitungen wie das Hinzufügen von Schönheitsfiltern zur Kamera durchzuführen.

Wenn Sie eine benutzerdefinierte Image-Eingangsquelle zur benutzerdefinierten Steuerung der Kamera verwenden (z. B. die Verwendung von Schönheitsfilter-Bibliotheken, die Kamerazugriff erfordern), ist das Broadcast-SDK nicht mehr für die Verwaltung der Kamera verantwortlich. Stattdessen ist die Anwendung dafür verantwortlich, den Lebenszyklus der Kamera korrekt zu handhaben. Lesen Sie die offizielle Plattformdokumentation darüber, wie Ihre Anwendung die Kamera verwalten soll.

## Android

Erstellen Sie nach dem Erstellen einer `DeviceDiscovery`-Sitzung eine Bildeingabequelle:

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

Diese Methode gibt ein `CustomImageSource` zurück, welche eine Image-Quelle ist, die von einem Standard-Android-Surface unterstützt wird. Die Unterklasse `SurfaceSource` kann in der Größe geändert und gedreht werden. Sie können auch ein `ImagePreviewView` erstellen, um eine Vorschau seines Inhalts anzuzeigen.

So rufen Sie das zugrundeliegende `Surface` ab:

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

Dieses `Surface` kann als Ausgabepuffer für Image-Produzenten wie Camera2, OpenGL ES und andere Bibliotheken verwendet werden. Der einfachste Anwendungsfall ist das direkte Zeichnen einer statischen Bitmap oder Farbe in den Canvas des Surface. Viele Bibliotheken (wie Schönheitsfilter-Bibliotheken) bieten jedoch eine Methode, mit der eine Anwendung ein externes `Surface` zum Rendern angeben kann. Sie können eine solche Methode verwenden, um dieses `Surface` an die Filterbibliothek zu übergeben, die es der Bibliothek ermöglicht, verarbeitete Frames für das Streamen der Broadcast-Sitzung auszugeben.

Dieses `CustomImageSource` kann in einen `LocalStageStream` verpackt und von der `StageStrategy` zurückgegeben werden, um es in einer `Stage` zu veröffentlichen.

## iOS

Erstellen Sie nach dem Erstellen einer `DeviceDiscovery`-Sitzung eine Bildeingabequelle:

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

Diese Methode gibt eine `IVSCustomImageSource` zurück, welche eine Image-Quelle ist, die es der Anwendung ermöglicht, `CMSampleBuffers` manuell abzusenden. Informationen zu unterstützten Pixelformaten finden Sie in der iOS-Broadcast-SDK-Referenz; ein Link zur aktuellsten Version befindet sich in den Versionshinweisen zu Amazon IVS für die neueste Broadcast-SDK-Version.

Die an die benutzerdefinierte Quelle gesendeten Proben werden auf die Stage gestreamt:

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

Verwenden Sie diese Methode zum Streamen von Videos in einem Rückruf. Wenn Sie beispielsweise die Kamera verwenden, kann die Anwendung jedes Mal, wenn ein neuer Beispielpuffer von einer `AVCaptureSession` erhalten wird, den Beispielpuffer an die benutzerdefinierte Image-

Quelle weiterleiten. Falls gewünscht, kann die Anwendung eine weitere Verarbeitung (wie einen Schönheitsfilter) anwenden, bevor sie das Beispiel an die benutzerdefinierte Image-Quelle absendet.

Dieses `IVSCustomImageSource` kann in einen `IVSLocalStageStream` verpackt und von der `IVSStageStrategy` zurückgegeben werden, um es in einer `Stage` zu veröffentlichen.

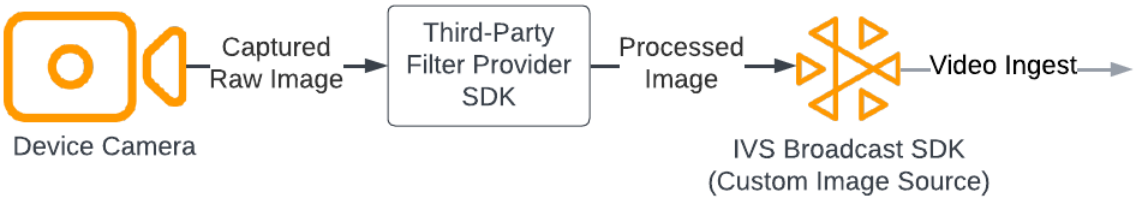
# IVS Broadcast SDK: Kamerafilter von Drittanbietern (Echtzeit-Streaming)

In diesem Handbuch wird davon ausgegangen, dass Sie bereits mit benutzerdefinierten Bild-Quellen vertraut sind und das Broadcast-SDK zu IVS-Echtzeit-Streaming in Ihre Anwendung integrieren.

Kamerafilter ermöglichen es Livestream-Erstellern, ihr Gesichts- oder Hintergrundbild zu verbessern oder zu ändern. Dies kann möglicherweise das Engagement der Zuschauer steigern, Zuschauer anlocken und das Live-Streaming-Erlebnis verbessern.

## Integration von Kamerafiltern von Drittanbietern

Sie können Kamera-Filter-SDKs von Drittanbietern in das IVS-Broadcast-SDK integrieren, indem Sie die Ausgabe des Filter-SDKs einer benutzerdefinierten Bildeingabequelle zuführen. Mit einer benutzerdefinierten Bildeingabequelle kann eine Anwendung ihre eigene Bildereingabe für das Broadcast SDK bereitstellen. Das SDK eines Drittanbieters von Filtern kann möglicherweise den Lebenszyklus der Kamera verwalten, um Bilder von der Kamera zu verarbeiten, einen Filtereffekt anzuwenden und in einem Format auszugeben, das an eine benutzerdefinierte Bildquelle übergeben werden kann.




Informationen zu integrierten Methoden zum Konvertieren eines Kamerarahmens mit angewendetem Filtereffekt in ein Format, das an eine benutzerdefinierte Bildeingabequelle übergeben werden kann, finden Sie in der Dokumentation Ihres Drittanbieters von Filtern. Der Prozess variiert, je nachdem, welche Version des IVS Broadcast SDK verwendet wird:

- Web – Der Filteranbieter muss in der Lage sein, seine Ausgabe auf einem Bildflächenelement zu rendern. Anschließend kann die Methode captureStream verwendet werden, um einen

MediaStream des Bildflächeninhalts zurückzugeben. Der MediaStream kann dann in eine Instance eines LocalStageStream umgewandelt und auf einer Stufe veröffentlicht werden.

- Android – Das SDK des Filteranbieters kann entweder einen Frame auf einem vom IVS-Broadcast-SDK bereitgestellten Android `Surface` rendern oder den Frame in eine Bitmap konvertieren. Wenn Sie eine Bitmap verwenden, kann diese dann durch Entsperren und Schreiben in eine Bildfläche in der zugrunde liegenden `Surface` gerendert werden, das von der benutzerdefinierten Bildquelle bereitgestellt wird.
- iOS – Das SDK eines Drittanbieter für Filter muss einen Kamerarahmen mit einem als `CMSampleBuffer` angewendeten Filtereffekt bereitstellen. Informationen dazu, wie Sie nach der Verarbeitung eines Kamerabilds ein `CMSampleBuffer` als endgültige Ausgabe erhalten, finden Sie in der SDK-Dokumentation Ihres Drittanbieters für Filter.

## BytePlus

### Android

Installieren und Einrichten des BytePlus-Effects-SDK

Einzelheiten zur Installation, Initialisierung und Einrichtung des BytePlus-Effects-SDK finden Sie im Android-Zugriffsleitfaden von BytePlus.

Einrichten der benutzerdefinierten Bildquelle

Führen Sie nach der Initialisierung des SDK verarbeitete Kamerarahmen mit einem Filtereffekt ein, der auf eine benutzerdefinierte Bildeingabequelle angewendet wird. Erstellen Sie dazu eine Instance eines `DeviceDiscovery`-Objekts sowie eine benutzerdefinierte Bildquelle. Beachten Sie, dass das Broadcast-SDK nicht mehr für die Verwaltung der Kamera verantwortlich ist, wenn Sie eine benutzerdefinierte Bildeingabequelle zur benutzerdefinierten Steuerung der Kamera verwenden. Stattdessen ist die Anwendung dafür verantwortlich, den Lebenszyklus der Kamera korrekt zu handhaben.

Java

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

```
var filterStream = ImageLocalStageStream(customSource)
```

Konvertieren der Ausgabe in ein Bitmap und Zuführen zu einer benutzerdefinierten Bildeingabequelle

Damit Kamerarahmen, auf die ein vom BytePlus-Effects-SDK angewendeter Filtereffekt angewendet wurde, direkt an das IVS-Broadcast-SDK weitergeleitet werden können, konvertieren Sie die Texturausgabe des BytePlus-Effects-SDK in eine Bitmap. Bei der Verarbeitung eines Bildes wird die `onDrawFrame()`-Methode vom SDK aufgerufen. Die `onDrawFrame()`-Methode ist eine öffentliche Methode der GLSurfaceView.Renderer-Schnittstelle von Android. In der von BytePlus bereitgestellten Android-Beispielanwendung wird diese Methode bei jedem Kamerarahmen aufgerufen. Diese gibt eine Textur aus. Gleichzeitig können Sie die `onDrawFrame()`-Methode mit Logik ergänzen, um diese Textur in eine Bitmap umzuwandeln und sie einer benutzerdefinierten Bildeingabequelle zuzuführen. Verwenden Sie für diese Konvertierung, wie im folgenden Codebeispiel gezeigt, die vom BytePlus-SDK bereitgestellte `transferTextureToBitmap`-Methode. Diese Methode wird von der com.bytedance.labcv.core.util.ImageUtil-Bibliothek aus dem BytePlus Effects SDK bereitgestellt, wie im folgenden Codebeispiel gezeigt. Anschließend können Sie auf das zugrunde liegende Android `Surface` des `CustomImageSource` rendern, indem Sie die resultierende Bitmap auf die Bildfläche einer Oberfläche schreiben. Viele aufeinanderfolgende Aufrufe von `onDrawFrame()` führen zu einer Folge von Bitmaps und erzeugen bei der Kombination einen Videostream.

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

Einzelheiten zur Integration des DeepAR-SDK in das Android-IVS-Broadcast-SDK finden Sie im Android-Integrationshandbuch von DeepAR.

## iOS

Einzelheiten zur Integration des DeepAR-SDK in das iOS IVS-Broadcast-SDK finden Sie im iOS-Integrationshandbuch von DeepAR.

# Snap

## Web

In diesem Abschnitt wird davon ausgegangen, dass Sie bereits mit dem Veröffentlichen und Abonnieren von Videos mithilfe des Web-Broadcast-SDK vertraut sind.

Um das Camera-Kit-SDK von Snap mit dem Web-Broadcast-SDK des IVS-Echtzeit-Streaming zu integrieren, müssen Sie Folgendes tun:

1. Installieren Sie das Camera-Kit-SDK und das Webpack. (In unserem Beispiel wird Webpack als Bundler verwendet, Sie können jedoch jeden Bundler Ihrer Wahl verwenden.)
2. Geben Sie einen Namen für den Benutzer ein und klicken Sie dann auf `index.html`.
3. Fügen Sie Einrichtungselemente hinzu.
4. Zeigen Sie Teilnehmer an und legen Sie sie fest.
5. Zeigen Sie die angeschlossenen Kameras und Mikrofone an.
6. Erstellen Sie eine Camera-Kit-Sitzung.
7. Rufen Sie ein Objektiv auf und wenden Sie es an.
8. Rendern Sie die Ausgabe einer Camera-Kit-Sitzung in einer Bildfläche.
9. Stellen Sie Camera Kit eine Medienquelle zum Rendern und Veröffentlichen eines `LocalStageStream` zur Verfügung.
10. Erstellen Sie eine Webpack-Konfigurationsdatei.

Jeder dieser Schritte wird nachfolgend beschrieben.

Installieren des Camera-Kit-SDK und des Webpacks

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

index.html erstellen

Erstellen Sie als Nächstes das HTML-Boilerplate und importieren Sie das Web-Broadcast-SDK als Skript-Tag. Stellen Sie im folgenden Code sicher, dass Sie <SDK version> durch die von Ihnen verwendete Broadcast-SDK-Version ersetzen.

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>
```

### Einrichtungselemente hinzufügen

Erstellen Sie den HTML-Code für die Auswahl einer Kamera und eines Mikrofons und die Angabe eines Teilnehmer-Tokens:

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>
```

Fügen Sie darunter zusätzlichen HTML-Code hinzu, um Kamera-Feeds von lokalen und entfernten Teilnehmern anzuzeigen:

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>
```

Laden Sie zusätzliche Logik, einschließlich Hilfsmethoden zum Einrichten der Kamera und der gebündelten JavaScript-Datei. (Später in diesem Abschnitt werden Sie diese JavaScript-Dateien erstellen und sie in einer einzigen Datei bündeln, damit Sie Camera Kit als Modul importieren können.

Die gebündelte JavaScript-Datei enthält die Logik, mit der Sie Camera Kit festlegen können, ein Objektiv anwenden und den Kamera-Feed mit einem Objektiv in einer Stufe veröffentlichen).

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>
```

Teilnehmer anzeigen und einrichten

Erstellen Sie als Nächstes `helpers.js`, das Hilfsmethoden zum Anzeigen und Einrichten von Teilnehmern enthält:

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;
}
```

### Angeschlossene Kameras und Mikrofone anzeigen

Erstellen Sie als Nächstes `media-devices.js`, das Hilfsmethoden zum Anzeigen von Kameras und Mikrofonen enthält, die mit Ihrem Gerät verbunden sind:

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,
    },
  });
}
```

### Erstellen einer Camera-Kit-Sitzung

Erstellen Sie `stages.js`, das die Logik zum Anwenden eines Objektivs auf den Kamera-Feed und zum Veröffentlichen des Feeds in einer Stufe enthält. Im ersten Teil dieser Datei wird das Broadcast-SDK und das Camera-Kit-Web-SDK importiert und die Variablen, die mit jedem SDK verwenden werden, initialisiert. Es wird eine Camera-Kit-Sitzung erstellt , durch Aufrufen von `createSession` nach dem Bootstrapping das Camera-Kit-Web-SDK. Beachten Sie, dass ein Elementobjekt einer Bildfläche an eine Sitzung übergeben wird. Dies weist Camera Kit an, in dieser Bildfläche zu rendern.

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 });
```

### Objektiv abrufen und anwenden

Um Ihre Objektive abzurufen, geben Sie Ihre Objektivgruppen-ID ein, die Sie im Camera-Kit-Entwicklerportal finden. Dieses Beispiel ist einfach gehalten, denn es wird das erste Objektiv aus dem zurückgegebenen Objektiv-Array verwendet.

JavaScript

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

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

Rendern der Ausgabe einer Camera-Kit-Sitzung in einer Bildfläche

Verwenden Sie die Methode captureStream, um einen `MediaStream` des Bildflächeninhalts zurückzugeben. Die Bildfläche enthält einen Videostream des Kamera-Feeds mit angewendetem Objektiv. Fügen Sie außerdem Ereignis-Listener für Schaltflächen zum Stummschalten von Kamera und Mikrofon sowie Ereignis-Listener für das Betreten und Verlassen einer Stufe hinzu. Im Ereignis-Listener für den Beitritt zu einer Stufe übergeben wir eine Camera-Kit-Sitzung und das `MediaStream` aus der Bildfläche, damit es in einer Stufe veröffentlicht werden kann.

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();
  });
};
```

Stellen Sie dem Camera Kit eine Medienquelle zum Rendern zur Verfügung und veröffentlichen Sie einen LocalStageStream

Um einen Videostream mit einem angewandten Objektiv zu veröffentlichen, rufen Sie eine Funktion mit dem Namen `setCameraKitSource` auf, um das zuvor von der Bildfläche erfasste `MediaStream` zu übergeben. Der `MediaStream` von der Bildfläche tut im Moment nichts, weil wir unseren lokalen Kamera-Feed noch nicht integriert haben. Wir können unseren lokalen Kamera-Feed integrieren, indem wir die Hilfsmethode `getCamera` aufrufen und sie `localCamera` zuweisen.

Wir können dann unseren lokalen Kamera-Feed (über `localCamera`) und das Sitzungsobjekt an `setCameraKitSource` übergeben. Die `setCameraKitSource`-Funktion konvertiert unseren lokalen Kamera-Feed in eine Medienquelle für CameraKit, indem sie `createMediaStreamSource` aufruft. Die Medienquelle für `CameraKit` wird dann umgewandelt, um die nach vorne gerichtete Kamera zu spiegeln. Der Objektiveffekt wird dann auf die Medienquelle angewendet und durch Aufrufen von `session.play()` auf die Ausgabe-Bildfläche gerendert.

Nachdem das Objektiv nun auf das von der Bildfläche erfasste `MediaStream` angewendet wurde, können wir es in einer Stufe veröffentlichen. Wir machen das, indem wir einen `LocalStageStream` mit den Videospuren des `MediaStream` erstellen. Eine Instance von `LocalStageStream` kann dann zur Veröffentlichung an eine `StageStrategy` übergeben werden.

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;
    },
  };
```

Der verbleibende Code unten dient der Erstellung und Verwaltung der Stufe:

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();
```

Erstellen einer Webpack-Konfigurationsdatei

Erstellen Sie `webpack.config.js` und fügen Sie den folgenden Code hinzu. Dadurch wird die oben genannte Logik gebündelt, sodass Sie die Importanweisung zur Verwendung von Camera Kit verwenden können.

JavaScript

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

Führen Sie schließlich `npm run build` aus, um Ihr JavaScript nach den Vorgaben in der Webpack-Konfigurationsdatei zu bündeln. Sie können anschließend HTML und JavaScript von einem Webserver bereitstellen. Sie können beispielsweise den HTTP-Server von Python verwenden und localhost:8000 öffnen, um das Ergebnis anzuzeigen:

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

## Android

Um das Camera-Kit-SDK von Snap mit dem IVS-Android-Broadcast-SDK zu integrieren, müssen Sie das Camera-Kit-SDK installieren, eine Camera-Kit-Sitzung initialisieren, ein Objektiv anwenden und die Ausgabe der Camera-Kit-Sitzung an die benutzerdefinierte Bildeingabequelle weiterleiten.

Um das Camera Kit SDK zu installieren, fügen Sie Folgendes zur Datei Ihres `build.gradle`-Moduls hinzu. Ersetzen Sie `$cameraKitVersion` durch die neueste Version des Camera-Kit-SDK.

Java

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

Initialisieren und beziehen Sie eine `cameraKitSession`. Camera Kit bietet auch einen praktischen Wrapper für die APIs des CameraX von Android, sodass Sie keine komplizierte

Logik schreiben müssen, um CameraX mit Camera Kit zu verwenden. Sie können das `CameraXImageProcessorSource`-Objekt als Quelle für ImageProcessor verwenden, wodurch Sie Streaming-Frames für die Kameravorschau starten können.

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();
   }
```

Objektive abrufen und anwenden

Sie können Objektive und ihre Reihenfolge im Karussell im Entwickler-Portal des Camera Kit konfigurieren:

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);
      }));
```

```
});
```

Senden Sie zur Übertragung verarbeitete Bilder an die zugrunde liegende `Surface` einer benutzerdefinierten Bildquelle. Verwenden Sie ein `DeviceDiscovery`-Objekt und erstellen Sie eine `CustomImageSource`, um ein `SurfaceSource` zurückzugeben. Anschließend können Sie die Ausgabe einer `CameraKit`-Sitzung in der `Surface` rendern die von der `SurfaceSource` bereitgestellt wird.

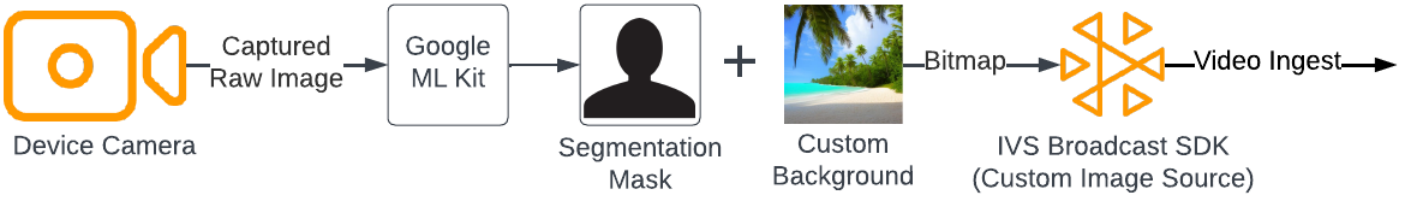
Java

```
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
```

## Ersetzen des Hintergrunds

Beim Ersetzen des Hintergrunds handelt es sich um eine Art Kamerafilter, mit dem Livestream-Ersteller ihren Hintergrund ändern können. Wie im folgenden Diagramm dargestellt, umfasst das Ersetzen Ihres Hintergrunds Folgendes:

1. Abrufen eines Kamerabildes aus dem Live-Kamera-Feed.
2. Segmentierung in Vordergrund- und Hintergrundkomponenten mithilfe des Google-ML-Sets.
3. Kombinieren der resultierenden Segmentierungsmaske mit einem benutzerdefinierten Hintergrundbild.
4. Weitergabe an eine benutzerdefinierte Bildquelle zur Übertragung.




## Web

In diesem Abschnitt wird davon ausgegangen, dass Sie bereits mit dem Veröffentlichen und Abonnieren von Videos mithilfe des Web-Broadcast-SDK vertraut sind.

Um den Hintergrund eines Live-Streams durch ein benutzerdefiniertes Bild zu ersetzen, verwenden Sie das Selfie-Segmentierungsmodell mit MediaPipe Image Segmenter. Hierbei handelt es sich um ein Machine Learning-Modell, das identifiziert, welche Pixel im Video-Frame im Vorder- oder Hintergrund liegen. Anschließend können Sie die Ergebnisse des Modells verwenden, um den Hintergrund eines Live-Streams zu ersetzen, indem Sie Vordergrundpixel aus dem Video-Feed in ein benutzerdefiniertes Bild kopieren, das den neuen Hintergrund darstellt.

Um die Ersetzung des Hintergrunds in das Web-Broadcast-SDK für IVS-Echtzeit-Streaming zu integrieren, müssen Sie Folgendes tun:

1. Installieren Sie MediaPipe und Webpack. (In unserem Beispiel wird Webpack als Bundler verwendet, Sie können jedoch jeden Bundler Ihrer Wahl verwenden.)
2. Geben Sie einen Namen für den Benutzer ein und klicken Sie dann auf `index.html`.
3. Fügen Sie Medienelemente hinzu.
4. Fügen Sie ein Skript-Tag hinzu.
5. Geben Sie einen Namen für den Benutzer ein und klicken Sie dann auf `app.js`.
6. Lädt ein benutzerdefiniertes Hintergrundbild.
7. Erstellen Sie eine Instance von `ImageSegmenter`.
8. Rendern Sie den Video-Feed in eine Bildfläche.
9. Erstellen Sie eine Logik zum Ersetzen des Hintergrunds.
10. Erstellen Sie eine Webpack-Konfigurationsdatei.
11. Bündeln Sie Ihre JavaScript-Datei.

MediaPipe und Webpack installieren

Installieren Sie zunächst die npm-Pakete `@mediapipe/tasks-vision` und `webpack`. Das folgende Beispiel verwendet Webpack als JavaScript-Bundler. Sie können bei Bedarf einen anderen Bundler verwenden.

JavaScript

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

Stellen Sie sicher, dass Sie auch Ihr `package.json` aktualisieren, um `webpack` als Entwickler-Skript anzugeben:

JavaScript

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

index.html erstellen

Erstellen Sie als Nächstes das HTML-Boilerplate und importieren Sie das Web-Broadcast-SDK als Skript-Tag. Stellen Sie im folgenden Code sicher, dass Sie `<SDK version>` durch die von Ihnen verwendete Broadcast-SDK-Version ersetzen.

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>
```

Medienelemente hinzufügen

Fügen Sie als Nächstes ein Videoelement und zwei Bildflächen-Elemente innerhalb des Tags des Hauptteils hinzu. Das Videoelement enthält Ihren Live-Kamera-Feed und wird als Eingabe für den MediaPipe Image Segmenter verwendet. Das erste Bildflächenelement wird verwendet, um eine Vorschau des zu übertragenden Feeds zu rendern. Das zweite Bildflächenelement wird zum Rendern des benutzerdefinierten Bildes verwendet, das als Hintergrund verwendet wird. Da die zweite Bildfläche mit dem benutzerdefinierten Bild nur als Quelle zum programmgesteuerten Kopieren von Pixeln von dort auf die endgültige Bildfläche verwendet wird, ist sie nicht sichtbar.

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>
```

Hinzufügen eines Skript-Tags

Fügen Sie ein Skript-Tag hinzu, um eine gebündelte JavaScript-Datei zu laden, die den Code für die Ersetzung des Hintergrunds enthält, und veröffentlichen Sie sie in einer Stufe:

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

Erstellen von app.js

Erstellen Sie als Nächstes eine JavaScript-Datei, um die Elementobjekte für die Bildfläche und Videoelemente abzurufen, die auf der HTML-Seite erstellt wurden. Importieren Sie die Module `ImageSegmenter` und `FilesetResolver`. Das Modul `ImageSegmenter` wird zur Durchführung der Segmentierungsaufgabe verwendet.

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";
```

Erstellen Sie als Nächstes eine Funktion mit dem Namen `init()`, um den MediaStream von der Kamera des Benutzers abzurufen und jedes Mal eine Rückruffunktion aufzurufen, wenn ein Kamerarahmen vollständig geladen ist. Fügen Sie Ereignis-Listener für die Schaltflächen zum Beitreten und Verlassen einer Stufe hinzu.

Beachten Sie, dass wir beim Beitritt zu einer Stufe eine Variable mit dem Namen `segmentationStream` übergeben. Hierbei handelt es sich um einen Video-Stream, der von einem Bildflächenelement erfasst wird und ein Vordergrundbild enthält, das dem benutzerdefinierten Bild, das den Hintergrund darstellt, überlagert ist. Später wird dieser benutzerdefinierte Stream zum Erstellen einer Instance eines `LocalStageStream` verwendet, die in einer Stufe veröffentlicht werden kann.

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();
  });
};
```

Ein benutzerdefiniertes Hintergrundbild laden

Fügen Sie unten in der `init`-Funktion Code hinzu, um eine Funktion mit dem Namen `initBackgroundCanvas` aufzurufen, die ein benutzerdefiniertes Bild aus einer lokalen Datei lädt und es in einer Bildfläche rendert. Diese Funktion wird im nächsten Schritt definiert. Ordnen Sie das von der Kamera des Benutzers abgerufene `MediaStream` dem Videoobjekt zu. Später wird dieses Videoobjekt an den Image Segmenter übergeben. Legen Sie außerdem eine Funktion mit dem Namen `renderVideoToCanvas` als Rückruffunktion fest, die immer dann aufgerufen wird, wenn ein Videobild vollständig geladen wurde. Diese Funktion wird in einem späteren Schritt definiert.

JavaScript

```
initBackgroundCanvas();

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

Wir implementieren die `initBackgroundCanvas`-Funktion, die ein Bild aus einer lokalen Datei lädt. In diesem Beispiel wird das Bild von einem Strandes als benutzerdefinierter Hintergrund verwendet. Die Bildfläche mit dem benutzerdefinierten Bild wird nicht angezeigt, da Sie sie mit den Vordergrundpixeln aus dem Bildflächenelement mit dem Kamera-Feed zusammenführen.

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);
  };
};
```

Erstellen einer Instance von ImageSegmenter

Erstellen Sie als Nächstes eine Instance von `ImageSegmenter`, die das Bild segmentiert und das Ergebnis als Maske zurückgibt. Verwenden Sie beim Erstellen einer Instance eines `ImageSegmenter` das Selfie-Segmentierungsmodell.

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,
  });
};
```

Rendern des Video-Feeds auf eine Bildfläche

Erstellen Sie als Nächstes die Funktion, die den Video-Feed auf das andere Bildflächenelement rendert. Das Video-Feed muss auf einer Bildfläche gerendert werden, damit mithilfe der Bildflächen-2D-API die Vordergrundpixel daraus extrahiert werden können. Dabei übergeben wir auch einen Videoframe an unsere `ImageSegmenter`-Instance und verwenden dabei die Methode segmentforVideo, um den Vordergrund vom Hintergrund im Videoframe zu segmentieren. Wenn die Methode segmentforVideo zurückgegeben wird, ruft sie die benutzerdefinierte Rückruffunktion `replaceBackground` auf, um den Hintergrund zu ersetzen.

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);
};
```

Logik zum Ersetzen des Hintergrunds erstellen

Erstellen Sie die `replaceBackground`-Funktion, die das benutzerdefinierte Hintergrundbild mit dem Vordergrund aus dem Kamera-Feed zusammenführt, um den Hintergrund zu ersetzen. Die Funktion ruft zunächst die zugrunde liegenden Pixeldaten des benutzerdefinierten Hintergrundbilds und des Video-Feeds von den beiden zuvor erstellten Bildflächenelementen ab. Anschließend durchläuft es die von `ImageSegmenter` bereitgestellte Maske, die angibt, welche Pixel im Vordergrund stehen. Beim Durchlaufen der Maske kopiert es selektiv Pixel, die den Kamera-Feed des Benutzers enthalten, in die entsprechenden Hintergrund-Pixeldaten. Sobald das erledigt ist, konvertiert es die endgültigen Pixeldaten, wobei der Vordergrund in den Hintergrund kopiert wird, und auf eine Bildfläche dargestellt wird.

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);
}
```

Als Referenz finden Sie hier die vollständige `app.js`-Datei, die die gesamte obige Logik enthält:

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();
```

Erstellen einer Webpack-Konfigurationsdatei

Fügen Sie diese Konfiguration zu Ihrer Webpack-Konfigurationsdatei hinzu, um `app.js` zu bündeln, damit die Importaufrufe funktionieren:

JavaScript

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

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

Bündeln Ihrer JavaScript-Dateien

```
npm run build
```

Starten Sie einen einfachen HTTP-Server aus dem Verzeichnis, das `index.html` enthält, und öffnen Sie `localhost:8000`, um das Ergebnis anzuzeigen:

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

## Android

Um den Hintergrund in Ihrem Livestream zu ersetzen, können Sie die Selfie-Segmentierungs-API von Google ML Kit verwenden. Die Selfie-Segmentierungs-API akzeptiert ein Kamerabild als Eingabe und gibt eine Maske zurück. Diese liefert für jedes Pixel des Bildes einen Konfidenzwert, der angibt, ob es sich im Vordergrund oder im Hintergrund befand. Basierend auf dem Konfidenzwert können Sie dann die entsprechende Pixelfarbe entweder aus dem Hintergrundbild oder dem Vordergrundbild abrufen. Dieser Vorgang wird fortgesetzt, bis alle Konfidenzwerte in der Maske überprüft wurden. Das Ergebnis ist ein neues Array von Pixelfarben, das Vordergrundpixel kombiniert mit Pixeln aus dem Hintergrundbild enthält.

Um die Ersetzung im Hintergrund mit dem Android-Broadcast-SDK für ISV-Echtzeit-Streaming zu integrieren, müssen Sie wie folgt vorgehen:

1. Installieren Sie die CameraX-Bibliotheken und das Google ML-Kit.
2. Initialisieren Sie Boilerplate-Variablen.
3. Erstellen Sie eine benutzerdefinierte Bildquelle.
4. Verwalten Sie Kamerarahmen.
5. Übergeben Sie Kamerarahmen an Google ML Kit.
6. Überlagern Sie den Vordergrund des Kamerarahmens mit Ihrem benutzerdefinierten Hintergrund.
7. Führen Sie das neue Bild einer benutzerdefinierten Bildquelle zu.

CameraX-Bibliotheken und Google ML Kit installieren

Verwenden Sie die CameraX-Bibliothek von Android, um Bilder aus dem Live-Kamera-Feed zu extrahieren. Um die CameraX-Bibliothek und das Google ML-Kit zu installieren, fügen Sie Folgendes zur `build.gradle`-Datei Ihres Moduls hinzu. Ersetzen Sie `${camerax_version}` und `${google_ml_kit_version}` durch die neueste Version der CameraX- bzw. Google-ML-Kit-Bibliotheken.

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}"
```

Importieren Sie die folgenden Bibliotheken:

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
```

Initialisieren von Boilerplate-Variablen

Initialisieren Sie eine Instance von `ImageAnalysis` und eine Instance eines `ExecutorService`:

Java

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

Initialisieren Sie eine Segmenter-Instance im STREAM_MODE:

Java

```
private val options =
        SelfieSegmenterOptions.Builder()
```

```
            .setDetectorMode(SelfieSegmenterOptions.STREAM_MODE)
            .build()

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

Erstellen einer benutzerdefinierten Image-Quelle

Erstellen Sie in der onCreate-Methode Ihrer Aktivität eine Instance eines DeviceDiscovery-Objekts sowie eine benutzerdefinierte Bildquelle. Die von der benutzerdefinierten Bildquelle bereitgestellte Surface erhält das endgültige Bild, wobei der Vordergrund über einem benutzerdefinierten Hintergrundbild liegt. Anschließend erstellen Sie mithilfe der benutzerdefinierten Bildquelle eine Instance von einem ImageLocalStageStream. Die Instance von einem ImageLocalStageStream (in diesem Beispiel filterStream benannt) kann dann in einer Stufe veröffentlicht werden. Anweisungen zum Einrichten einer Stufe finden Sie im Handbuch des IVS-Android-Broadcast-SDK. Erstellen Sie abschließend auch einen Thread, der zur Verwaltung der Kamera verwendet wird.

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()
```

Verwalten von Kamerarahmen

Erstellen Sie als Nächstes eine Funktion zum Initialisieren der Kamera. Diese Funktion nutzt die CameraX-Bibliothek, um Bilder aus dem Live-Kamera-Feed zu extrahieren. Zunächst erstellen Sie eine Instance eines ProcessCameraProvider mit dem Namen cameraProviderFuture. Dieses Objekt stellt ein zukünftiges Ergebnis der Beschaffung eines Kameraanbieters dar. Anschließend laden Sie ein Bild aus Ihrem Projekt als Bitmap. In diesem Beispiel wird ein Bild von einem Strand als Hintergrund verwendet, es kann sich aber auch um ein beliebiges Bild handeln.

Anschließend fügen Sie einen Listener zu cameraProviderFuture hinzu. Dieser Listener wird benachrichtigt, wenn die Kamera verfügbar wird oder wenn beim Abrufen eines Kameraanbieters ein Fehler auftritt.

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))
    }
```

Erstellen Sie im Listener `ImageAnalysis.Builder`, um über den Live-Kamera-Feed auf jedes einzelne Bild zuzugreifen. Legen Sie die Gegendruckstrategie auf `STRATEGY_KEEP_ONLY_LATEST` fest. Dadurch wird sichergestellt, dass jeweils nur ein Kamerarahmen zur Verarbeitung geliefert wird. Konvertieren Sie jedes einzelne Kamerarahmen in eine Bitmap, sodass Sie dessen Pixel extrahieren und später mit dem benutzerdefinierten Hintergrundbild kombinieren können.

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())
```

Übergabe von Kamerarahmen an Google ML Kit

Erstellen Sie als Nächstes ein `InputImage` und übergeben Sie es zur Verarbeitung an die Segmenter-Instance. Ein `InputImage` kann aus einem `ImageProxy` erstellt werden, der von der Instance von `ImageAnalysis` bereitgestellt wird. Sobald dem Segmenter ein `InputImage` zur Verfügung gestellt wird, gibt er eine Maske mit Konfidenzwerten zurück, die die Wahrscheinlichkeit angeben, dass sich ein Pixel im Vordergrund oder Hintergrund befindet. Diese Maske bietet auch Breiten- und Höheneigenschaften, mit denen Sie ein neues Array erstellen, das die Hintergrundpixel des zuvor geladenen benutzerdefinierten Hintergrundbilds enthält.

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)
```

Überlagern des Vordergrunds des Kamerarahmens mit Ihrem benutzerdefinierten Hintergrund

Mit der Maske, die die Konfidenzwerte enthält, dem Kamerarahmen als Bitmap und den Farbpixeln aus dem benutzerdefinierten Hintergrundbild haben Sie alles, was Sie brauchen, um den Vordergrund über Ihren benutzerdefinierten Hintergrund zu legen. Die `overlayForeground`-Funktion wird dann mit folgenden Parametern aufgerufen:

Java

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

Diese Funktion durchläuft die Maske und prüft die Konfidenzwerte, um festzustellen, ob die entsprechende Pixelfarbe aus dem Hintergrundbild oder dem Kamerarahmen abgerufen werden soll. Wenn der Konfidenzwert angibt, dass sich ein Pixel in der Maske höchstwahrscheinlich im Hintergrund befindet, erhält er die entsprechende Pixelfarbe aus dem Hintergrundbild. Andernfalls wird die entsprechende Pixelfarbe vom Kamerarahmen abgerufen, um den Vordergrund zu erstellen. Sobald die Funktion die Iteration durch die Maske abgeschlossen hat, wird unter Verwendung des neuen Arrays von Farbpixeln eine neue Bitmap erstellt und zurückgegeben. Diese neue Bitmap enthält den Vordergrund, der sich mit dem benutzerdefinierten Hintergrund überlagert.

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
        )
    }
```

Das neue Bild einer benutzerdefinierten Bildquelle zuführen

Anschließend können Sie die neue Bitmap in die `Surface` schreiben, das von einer benutzerdefinierten Bildquelle bereitgestellt wird. Dadurch wird es in Ihre Stufe übertragen.

Java

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

Hier ist die vollständige Funktion zum Abrufen der Kamerarahmen, zum Übergeben an Segmenter und zum Überlagern mit dem Hintergrund:

Java

```
@androidx.annotation.OptIn(androidx.camera.core.ExperimentalGetImage::class)
```

```
    private fun startCamera(surface: Surface) {
        val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
        val imageResource = R.drawable.clouds
        val bgBitmap: Bitmap = BitmapFactory.decodeResource(resources, imageResource)
        var resultBitmap: Bitmap;

        cameraProviderFuture.addListener({
            // Used to bind the lifecycle of cameras to the lifecycle owner
            val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

            val imageAnalyzer = ImageAnalysis.Builder()
            analysisUseCase = imageAnalyzer
                .setTargetResolution(Size(720, 1280))
                .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
                .build()

            analysisUseCase!!.setAnalyzer(cameraExecutor) { imageProxy: ImageProxy ->
                val mediaImage = imageProxy.image
                val tempBitmap = imageProxy.toBitmap();
                val inputBitmap =
tempBitmap.rotate(imageProxy.imageInfo.rotationDegrees.toFloat())

                if (mediaImage != null) {
                    val inputImage =
                        InputImage.fromMediaImage(mediaImage,
imageProxy.imageInfo.rotationDegrees)

                    segmenter.process(inputImage)
                        .addOnSuccessListener { segmentationMask ->
                            val mask = segmentationMask.buffer
                            val maskWidth = segmentationMask.width
                            val maskHeight = segmentationMask.height
                            val backgroundPixels = IntArray(maskWidth * maskHeight)
                            bgBitmap.getPixels(backgroundPixels, 0, maskWidth, 0, 0,
maskWidth, maskHeight)

                            resultBitmap = overlayForeground(mask, maskWidth,
maskHeight, inputBitmap, backgroundPixels)
                            canvas = surface.lockCanvas(null);
                            canvas.drawBitmap(resultBitmap, 0f, 0f, null)

                            surface.unlockCanvasAndPost(canvas);

                        }
```

```
                        .addOnFailureListener { exception ->
                            Log.d("App", exception.message!!)
                        }
                        .addOnCompleteListener {
                            imageProxy.close()
                        }

                }
            };

            val cameraSelector = CameraSelector.DEFAULT_FRONT_CAMERA

            try {
                // Unbind use cases before rebinding
                cameraProvider.unbindAll()

                // Bind use cases to camera
                cameraProvider.bindToLifecycle(this, cameraSelector, analysisUseCase)

            } catch(exc: Exception) {
                Log.e(TAG, "Use case binding failed", exc)
            }

        }, ContextCompat.getMainExecutor(this))
    }
```

# IVS Broadcast SDK: Mobile Audiomodi (Echtzeit-Streaming)

Die Audioqualität ist ein wichtiger Bestandteil jedes Medienerlebnisses im echten Team, und es gibt keine einheitliche Audiokonfiguration, die für jeden Anwendungsfall am besten funktioniert. Um sicherzustellen, dass Ihre Benutzer beim Anhören eines IVS-Echtzeit-Streams über das beste Erlebnis verfügen, bieten unsere mobilen SDKs mehrere voreingestellte Audiokonfigurationen sowie bei Bedarf leistungsstärkere Anpassungen.

## Einführung

Die IVS-SDKs für mobile Übertragungen stellen eine `StageAudioManager`-Klasse bereit. Diese Klasse ist als zentraler Ansprechpartner für die Steuerung der zugrunde liegenden Audiomodi auf beiden Plattformen konzipiert. Unter Android steuert dies den AudioManager, einschließlich Audiomodus, Audioquelle, Inhaltstyp, Nutzung und Kommunikationsgeräte. Unter iOS steuert es die Anwendung AVAudioSession und ob VoiceProcessing aktiviert ist.

Wichtig: Interagieren Sie nicht mit `AVAudioSession` oder `AudioManager` direkt, während das IVS-Echtzeit-Broadcast-SDK aktiv ist. Dies kann dazu führen, dass das Audio verloren geht oder Audio vom falschen Gerät aufgenommen oder wiedergegeben wird.

Bevor Sie Ihr erstes `DeviceDiscovery`- oder `Stage`-Objekt erstellen, muss die `StageAudioManager`-Klasse konfiguriert werden.

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
```

Wenn nichts auf `StageAudioManager` festgelegt wird, bevor eine `DeviceDiscovery`- oder `Stage`-Instance initialisiert wird, wird automatisch die Voreinstellung `VideoChat` angewendet.

## Voreinstellungen für den Audio-Modus

Das Echtzeit-Broadcast-SDK bietet drei Voreinstellungen, die jeweils auf gängige Anwendungsfälle zugeschnitten sind, wie unten beschrieben. Für jede Voreinstellung werden fünf Hauptkategorien behandelt, die die Voreinstellungen voneinander unterscheiden.

### Video-Chat

Dies ist die Standardvoreinstellung, die für den Fall konzipiert ist, dass das lokale Gerät ein Echtzeitgespräch mit anderen Teilnehmern führen soll.

| Kategorie | Android | iOS |
| --- | --- | --- |
| Echounterdrückung | Aktiviert | Aktiviert |
| Lautstärkenregler | Anruflautstärke | Anruflautstärke |
| Auswahl des Mikrofons | Je nach Betriebssystem begrenzt. USB-Mikrofone sind möglicher weise nicht verfügbar. | Je nach Betriebssystem begrenzt. USB- und Bluetooth-Mikrofon e sind möglicherweise nicht verfügbar.<br><br>Bluetooth-Headsets, die sowohl die Ein- als auch die Ausgabe gemeinsam verarbeiten, sollten funktionieren; z. B. AirPods. |
| Audioausgabe | Jedes Ausgabegerät sollte funktionieren. | Je nach Betriebssystem begrenzt. Kabelgebundene Headsets sind möglicherweise nicht verfügbar. |
| Audioqualität | Mittel/Niedrig. Es hört sich wie ein Telefonanruf an, nicht wie die Medienwiedergabe. | Mittel/Niedrig. Es hört sich wie ein Telefonanruf an, nicht wie die Medienwiedergabe. |

## Nur Abonnement

Diese Voreinstellung ist für den Fall konzipiert, dass Sie andere Veröffentlichungsteilnehmer abonnieren, aber nicht selbst veröffentlichen möchten. Der Schwerpunkt liegt auf der Audioqualität und der Unterstützung aller verfügbaren Ausgabegeräte.

| Kategorie | Android | iOS |
| --- | --- | --- |
| Echounterdrückung | Disabled | Disabled |
| Lautstärkenregler | Medienlautstärke | Medienlautstärke |

| Kategorie | Android | iOS |
| --- | --- | --- |
| Auswahl des Mikrofons | Nicht verfügbar, diese Voreinstellung ist nicht für die Veröffentlichung konzipiert. | Nicht verfügbar, diese Voreinstellung ist nicht für die Veröffentlichung konzipiert. |
| Audioausgabe | Jedes Ausgabegerät sollte funktionieren. | Jedes Ausgabegerät sollte funktionieren. |
| Audioqualität | Hoch. Jeder Medientyp sollte klar zu hören sein, auch Musik. | Hoch. Jeder Medientyp sollte klar zu hören sein, auch Musik. |

## Studio

Diese Voreinstellung ist für qualitativ hochwertige Abonnements bei gleichzeitiger Beibehaltung der Möglichkeit zur Veröffentlichung konzipiert. Es erfordert, dass die Aufnahme- und Wiedergabe-Hardware eine Echounterdrückung ermöglicht. Ein Anwendungsfall hierfür wäre die Verwendung eines USB-Mikrofons und eines kabelgebundenen Headsets. Das SDK sorgt für die höchste Audioqualität und verlässt sich dabei auf die physische Trennung dieser Geräte, um Echos zu vermeiden.

| Kategorie | Android | iOS |
| --- | --- | --- |
| Echounterdrückung | Disabled | Disabled |
| Lautstärkenregler | Medienlautstärke in den meisten Fällen. Anruflautstärke, wenn ein Bluetooth-Mikrofon angeschlossen ist. | Medienlautstärke |
| Auswahl des Mikrofons | Jedes Mikrofon sollte funktionieren. | Jedes Mikrofon sollte funktionieren. |
| Audioausgabe | Jedes Ausgabegerät sollte funktionieren. | Jedes Ausgabegerät sollte funktionieren. |
| Audioqualität | Hoch. Beide Seiten sollten in der Lage sein, Musik zu senden und | Hoch. Beide Seiten sollten in der Lage sein, Musik zu senden und |

| Kategorie | Android | iOS |
| --- | --- | --- |
| | sie auf der anderen Seite klar zu hören.<br><br>Wenn ein Bluetooth-Headset angeschlossen ist, nimmt die Audioqualität ab, da der Bluetooth-SCO-Modus aktiviert ist. | sie auf der anderen Seite klar zu hören.<br><br>Wenn ein Bluetooth-Headset angeschlossen ist, kann es je nach Headset aufgrund der Aktivierung des Bluetooth SCO-Modus zu einer Verschlechterung der Audioqualität kommen. |

## Fortschrittliche Anwendungsfälle

Über die Voreinstellungen hinaus ermöglichen sowohl die Echtzeit-Streaming-Broadcast-SDKs für iOS als auch Android die Konfiguration der zugrunde liegenden Audiomodi der Plattform:

- Legen Sie unter Android AudioSource, Usage und ContentType fest.
- Verwenden Sie unter iOS AVAudioSession.Category, AVAudioSession.CategoryOptions, AVAudioSession.Mode und die Möglichkeit, beim Veröffentlichen umzuschalten, ob die Sprachverarbeitung aktiviert ist oder nicht.

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
```

### Veröffentlichen mit Bluetooth unter Android

Das SDK kehrt automatisch zur Voreinstellung `VIDEO_CHAT` unter Android zurück, wenn die folgenden Bedingungen erfüllt sind:

- Die zugewiesene Konfiguration verwendet den Nutzungswert `VOICE_COMMUNICATION` nicht.
- Ein Bluetooth-Mikrofon ist mit dem Gerät verbunden.
- Der lokale Teilnehmer veröffentlicht in einer Stufe.

Hierbei handelt es sich um eine Einschränkung des Android-Betriebssystems hinsichtlich der Verwendung von Bluetooth-Headsets zur Audioaufzeichnung.

## Integration mit anderen SDKs

Da sowohl iOS als auch Android nur einen aktiven Audiomodus pro Anwendung unterstützen, kommt es häufig zu Konflikten, wenn Ihre Anwendung mehrere SDKs verwendet, die die Steuerung des Audiomodus erfordern. Wenn Sie auf solche Konflikte stoßen, können Sie einige gängige Lösungsstrategien ausprobieren, die nachfolgend erläutert werden.

### Werte im Audiomodus anpassen

Mithilfe der erweiterten Audiokonfigurationsoptionen des IVS-SDK oder der Funktionalität des anderen SDK können Sie die beiden SDKs auf die zugrunde liegenden Werte ausrichten.

## Agora

### iOS

Wenn Sie unter iOS dem Agora SDK mitteilen, dass die `AVAudioSession` aktiv bleiben soll, wird verhindert, dass es deaktiviert wird, während das Broadcast-SDK für IVS-Echtzeit-Streaming es verwendet.

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

### Android

Vermeiden Sie den Aufruf von `setEnableSpeakerphone` in `RtcEngine` und rufen Sie `enableLocalAudio(false)` während der Veröffentlichung mit dem Broadcast-SDK für IVS-Echtzeit-Streaming auf. Sie können `enableLocalAudio(true)` erneut aufrufen, wenn das IVS-SDK keine Veröffentlichung durchführt.

# Amazon EventBridge mit IVS-Echtzeit-Streaming verwenden

Sie können Amazon EventBridge verwenden, um Ihre Amazon Interactive Video Service (IVS) Streams zu überwachen.

Amazon IVS sendet Änderungsereignisse zum Status Ihrer Streams an Amazon EventBridge. Alle bereitgestellten Ereignisse sind gültig. Allerdings werden Ereignisse auf Best-Effort-Basis gesendet, was bedeutet, dass keine Garantie für Folgendes besteht:

- Ereignisse werden übermittelt – Ein festgelegtes Ereignis kann auftreten (z. B. ein Stream startet), aber es ist möglich, dass Amazon IVS kein entsprechendes Ereignis an EventBridge sendet. Amazon IVS versucht, Ereignisse mehrere Stunden vor dem Aufgeben zu liefern.
- Ereignisse, die geliefert werden, kommen in einem bestimmten Zeitrahmen an – Sie können Ereignisse erhalten, die bis zu ein paar Stunden alt sind.
- Ereignisse werden in der richtigen Reihenfolge geliefert – Ereignisse können ungeordnet sein, insbesondere wenn sie innerhalb kurzer Zeit zueinander gesendet werden. Beispielsweise könnte „Teilnehmer nicht veröffentlicht“ vor „Teilnehmer veröffentlicht“ angezeigt werden.

Obwohl es selten ist, dass Ereignisse fehlen, spät oder nicht in richtiger Reihenfolge sind, sollten Sie diese Möglichkeiten berücksichtigen, wenn Sie geschäftskritische Programme schreiben, die von der Reihenfolge oder dem Vorhandensein von Benachrichtigungsereignissen abhängen.

Sie können EventBridge für jedes der folgenden Ereignisse erstellen.

| Ereignistyp | Veranstaltung | Gesendet, wenn ... |
|---|---|---|
| Status der IVS-Zusammensetzung | Zielfehler | Ein Ausgabeversuch an ein Ziel ist fehlgeschlagen. Beispielsweise ist die Übertragung an einen Kanal fehlgeschlagen, weil kein Stream-Schlüssel vorhanden war oder eine andere Übertragung stattfand. |
| Status der IVS-Zusammensetzung | Zielstart | Die Ausgabe an ein Ziel wurde erfolgreich gestartet. |

| Ereignistyp | Veranstaltung | Gesendet, wenn ... |
|---|---|---|
| Status der IVS-Zusammensetzung | Zielende | Die Ausgabe an ein Ziel ist abgeschlossen. |
| Status der IVS-Zusammensetzung | Wiederverbindung zum Ziel | Die Ausgabe an ein Ziel wurde unterbrochen. Es wird versucht, die Verbindung wiederherzustellen. |
| Status der IVS-Zusammensetzung | Beginn der Sitzung | Es wurde eine Sitzung zur Zusammensetzung erstellt. Dieses Ereignis wird ausgelöst, wenn eine Zusammensetzungsprozess-Pipeline erfolgreich initialisiert wurde. Zu diesem Zeitpunkt hat die Zusammensetzungs-Pipeline erfolgreich eine Stufe abonniert, empfängt Medien und kann Videos erstellen. |
| Status der IVS-Zusammensetzung | Ende der Sitzung | Eine Zusammensetzungs-Sitzung ist abgeschlossen. |
| Status der IVS-Zusammensetzung | Sitzungsfehler | Eine Zusammensetzungs-Pipeline konnte nicht initialisiert werden, weil die Stage-Ressourcen nicht verfügbar waren oder ein anderer interner Fehler vorliegt. |
| IVS-Bühne-Aktualisierung | Teilnehmer Published | Ein Teilnehmer beginnt, zu einer Bühne zu veröffentlichen. |
| IVS-Bühne-Aktualisierung | Teilnehmer Unveröffentlicht | Ein Teilnehmer hat aufgehört, zu einer Bühne zu veröffentlichen. |

# Erstellen von Amazon EventBridge Regeln für Amazon IVS

Sie können eine Regel erstellen, die bei einem von Amazon IVS ausgegebenen Ereignis ausgelöst wird. Folgen Sie den Schritten in Erstellen Sie eine Regel in Amazon

EventBridge im Benutzerhandbuch für Amazon EventBridge. Wählen Sie bei der Auswahl eines Services Interactive Video Service (IVS).

## Beispiele: Status der Zusammensetzung

Zielfehler: Dieses Ereignis wird gesendet, wenn ein Versuch zur Ausgabe an ein Ziel fehlgeschlagen ist. Beispielsweise ist die Übertragung an einen Kanal fehlgeschlagen, weil kein Stream-Schlüssel vorhanden war oder eine andere Übertragung stattfand.

```
{
   "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"
   }
}
```

Zielstart: Dieses Ereignis wird gesendet, wenn die Ausgabe an ein Ziel erfolgreich gestartet wurde.

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

Zielende: Dieses Ereignis wird gesendet, wenn die Ausgabe an ein Ziel abgeschlossen ist.

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

Wiederverbindung zum Ziel: Dieses Ereignis wird gesendet, wenn die Ausgabe an ein Ziel unterbrochen wurde und versucht wird, die Verbindung wiederherzustellen.

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

Sitzungsstart: Dieses Ereignis wird gesendet, wenn eine Zusammensetzungssitzung erstellt wurde. Dieses Ereignis wird ausgelöst, wenn eine Zusammensetzungsprozess-Pipeline erfolgreich initialisiert wurde. Zu diesem Zeitpunkt hat die Zusammensetzungs-Pipeline erfolgreich eine Stufe abonniert, empfängt Medien und kann Videos erstellen.

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

Sitzungsende: Dieses Ereignis wird gesendet, wenn eine Zusammensetzungssitzung abgeschlossen ist und alle Ressourcen gelöscht wurden.

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

Sitzungsfehler: Dieses Ereignis wird gesendet, wenn die Initialisierung einer Zusammensetzungs-Pipeline fehlschlägt, weil die Stufenressourcen nicht verfügbar sind, keine Teilnehmer in der Stufe sind oder ein anderer interner Fehler vorliegt.

```
{
  "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"
  }
}
```

# Beispiele: Bühne-Aktualisierung

Zu den Bühne-Aktualisierung-Ereignissen gehören ein Ereignisname (der das Ereignis klassifiziert) und Metadaten zum Ereignis. Zu den Metadaten gehören die Teilnehmer-ID, die das Ereignis ausgelöst hat, die zugehörigen Bühnen- und Sitzungs-IDs sowie die Benutzer-ID.

Teilnehmer Veröffentlicht: Dieses Ereignis wird gesendet, wenn ein Teilnehmer beginnt, zu einer Bühne zu veröffentlichen.

```
{
    "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"
    }
}
```

Teilnehmer Unveröffentlicht: Dieses Ereignis wird gesendet, wenn ein Teilnehmer damit aufgehört hat, zu einer Bühne zu veröffentlichen.

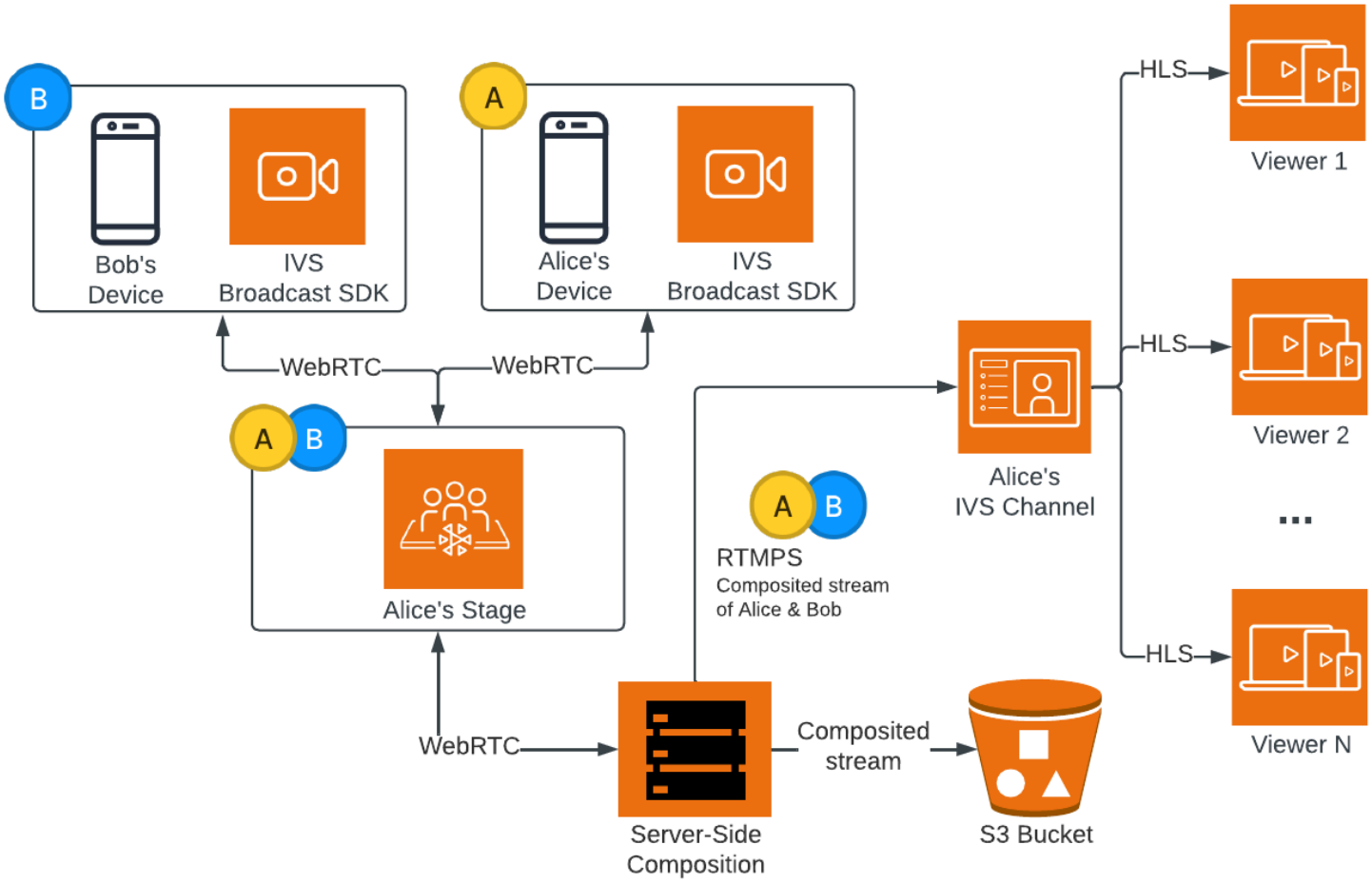
```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"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

# Serverseitige Zusammensetzung (Echtzeit-Streaming)

Die serverseitige Zusammensetzung verwendet einen IVS-Server, um Audio- und Videodaten von allen Teilnehmern der Stufe zu mischen und sendet dieses gemischte Video dann an einen IVS-Kanal (z. B. um ein größeres Publikum zu erreichen) oder einen S3-Bucket. Die serverseitige Zusammensetzung wird über IVS-Steuerebenen-Endpunkte in der Heimatregion der Stufe aufgerufen.

Die Übertragung oder Aufzeichnung einer Stufe mittels serverseitiger Zusammensetzung bietet zahlreiche Vorteile und ist daher eine attraktive Wahl für Benutzer, die effiziente und zuverlässige cloudbasierte Video-Workflows suchen.

Dieses Diagramm veranschaulicht, wie serverseitige Zusammensetzung funktioniert:



# Vorteile

Im Vergleich zur clientseitigen Zusammensetzung bietet die serverseitige Zusammensetzung die folgenden Vorteile:

- Reduzierte Client-Last – Bei der serverseitigen Zusammensetzung wird die Last der Verarbeitung und Kombination von Audio- und Videoquellen von einzelnen Client-Geräten auf den Server selbst verlagert. Durch die serverseitige Zusammensetzung entfällt die Anforderung, dass Client-Geräte ihre CPU- und Netzwerkressourcen für die Zusammensetzung der Anzeige und deren Übertragung an IVS verwenden. Dies bedeutet, dass Zuschauer die Übertragung ansehen können, ohne dass ihre Geräte ressourcenintensive Aufgaben erledigen müssen, was zu einer verbesserten Akkulaufzeit und einem flüssigeren Seherlebnis führen kann.
- Gleichbleibende Qualität – Die serverseitige Zusammensetzung ermöglicht eine präzise Kontrolle über die Qualität, Auflösung und Bitrate des endgültigen Streams. Dies gewährleistet ein einheitliches Seherlebnis für alle Zuschauer, unabhängig von den Fähigkeiten ihrer einzelnen Geräte.
- Ausfallsicherheit – Durch die Zentralisierung des Zusammensetzungsprozesses auf dem Server wird die Übertragung stabiler. Selbst wenn ein Publisher-Gerät technischen Einschränkungen oder Schwankungen unterliegt, kann sich der Server anpassen und allen Zuschauern einen reibungsloseren Stream bieten.
- Bandbreiteneffizienz – Da der Server die Zusammensetzung übernimmt, müssen Stufen-Publisher keine zusätzliche Bandbreite für die Übertragung des Videos an IVS aufwenden.

Um eine Stufe an einen IVS-Kanal zu übertragen, können Sie alternativ die Zusammensetzung clientseitig durchführen; Weitere Informationen finden Sie unter Aktivierung mehrerer Hosts auf einem IVS-Stream im Benutzerhandbuch zum IVS-Streaming mit niedriger Latenz.

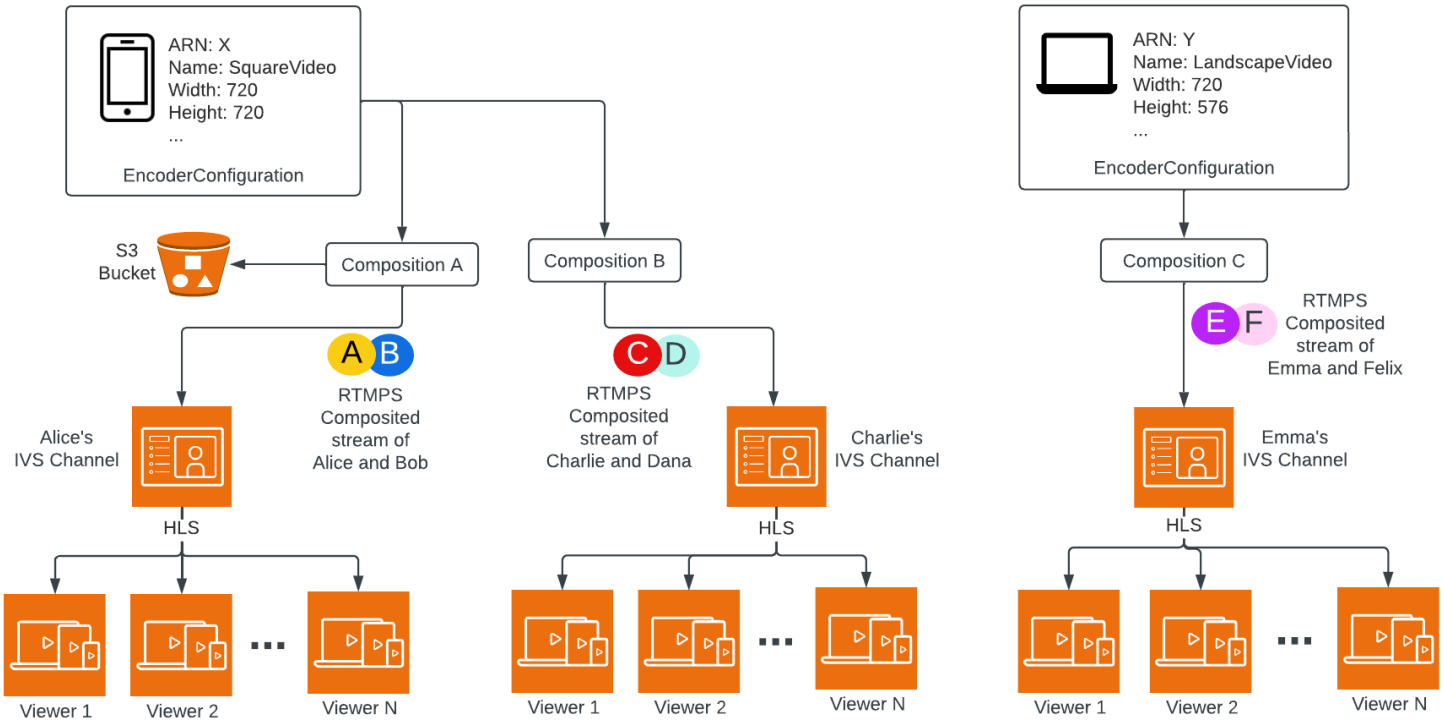
# IVS-API

Die serverseitige Zusammensetzung verwendet diese wichtigen API-Elemente:

- Mit einem EncoderConfiguration-Objekt können Sie das Format des zu generierenden Videos anpassen (Höhe, Breite, Bitrate und andere Streaming-Parameter). Sie können eine EncoderConfiguration bei jedem Aufruf des StartComposition-Endpunkts wiederverwenden.
- Zusammensetzungs-Endpunkte verfolgen die Video-Zusammensetzung und die Ausgabe an einem IVS-Kanal.

- StorageConfiguration verfolgt den S3-Bucket, in dem Zusammensetzungen aufgezeichnet werden.

Um die serverseitige Zusammensetzung zu verwenden, müssen Sie eine EncoderConfiguration erstellen und diese beim Aufruf des StartComposition-Endpunkts anfügen. In diesem Beispiel wird die SquareVideo EncoderConfiguration in zwei Zusammensetzungen verwendet:




Vollständige Informationen finden Sie unter API-Referenz zu IVS-Echtzeit-Streaming.

## Layouts

Standardmäßig verwendet das Feature für die serverseitige Zusammensetzung ein Raster-Layout, um die Teilnehmer der Stufe in gleich großen Slots anzuordnen:

Dieses Layout bietet Kunden die Möglichkeit, einen hervorgehobenen Slot zu konfigurieren und aufzurufen. Der hervorgehobene Slot befindet sich auf dem Hauptbildschirm, andere Teilnehmer werden in gleich großen Slots darunter angezeigt:


Hinweis: Die maximale Auflösung, die von einem Stufen-Publisher bei serverseitiger Zusammensetzung unterstützt wird, beträgt 1080 p. Wenn ein Publisher Videos mit einer höheren Auflösung als 1080 p sendet, wird der Publisher als reiner Audio-Teilnehmer gerendert.

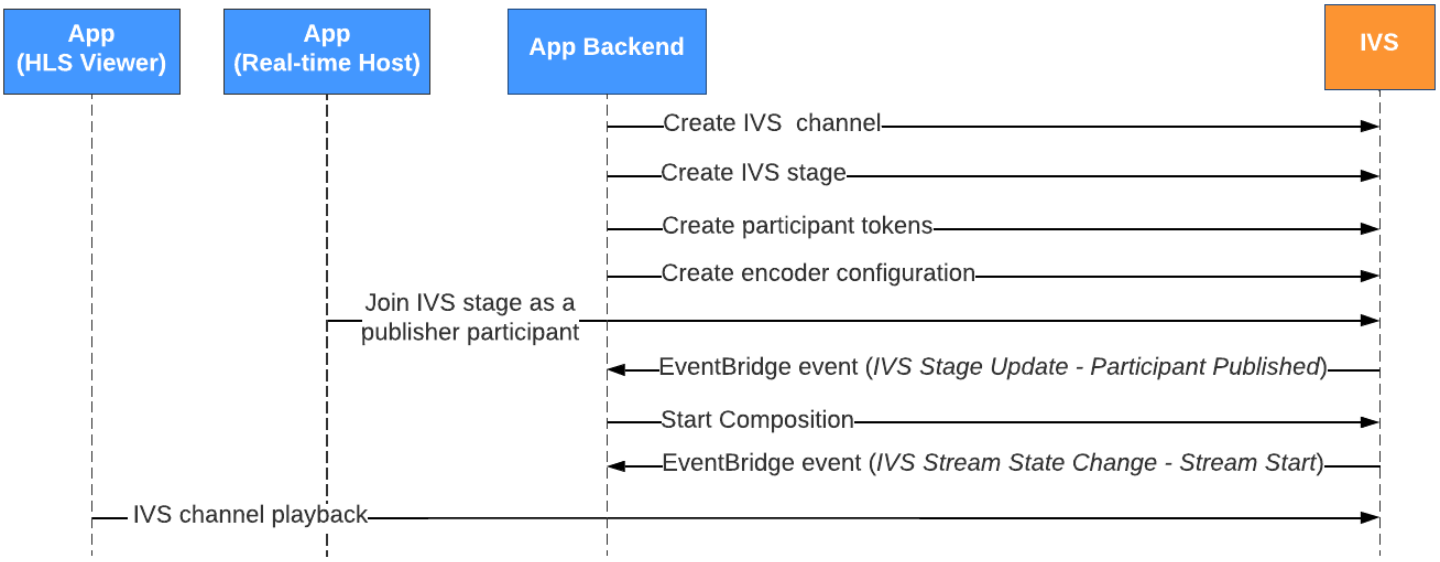
# Erste Schritte

## Voraussetzungen

Um die serverseitige Zusammensetzung verwenden zu können, müssen Sie über eine Stufe mit aktiven Publishern verfügen und einen IVS-Kanal und/oder einen S3-Bucket als Zusammensetzungsziel verwenden. Nachfolgend wird ein möglicher Workflow beschrieben,

der EventBridge-Ereignisse verwendet, um eine Zusammensetzung zu starten, die die Stufe an einen IVS-Kanal sendet, wenn ein Teilnehmer etwas veröffentlicht. Alternativ können Sie Zusammensetzungen basierend auf Ihrer eigenen App-Logik starten und stoppen. Unter Zusammengesetzte Aufzeichnung finden Sie ein weiteres Beispiel, das die Verwendung serverseitiger Zusammensetzung zur direkten Aufzeichnung einer Phase in einem S3-Bucket demonstriert.

1. Erstellen Sie einen IVS-Kanal. Weitere Informationen finden Sie unter Erste Schritte mit Amazon-IVS-Streaming mit niedriger Latenz.
2. Erstellen Sie für jeden Publisher eine IVS-Stufe und Teilnehmer-Tokens.
3. Erstellen Sie eine EncoderConfiguration.
4. Treten Sie der Stufe bei und veröffentlichen Sie dort. (Weitere Informationen finden Sie in den Abschnitten „Veröffentlichen und Abonnieren“ der SDK-Anleitungen für Echtzeit-Streaming-Broadcasts: Web, Android und iOS.)
5. Wenn Sie ein vom Teilnehmer veröffentlichtes EventBridge-Ereignis erhalten, rufen Sie StartComposition auf.
6. Warten Sie einige Sekunden und sehen Sie sich die zusammengesetzte Ansicht in der Kanalwiedergabe an.




Hinweis: Eine Zusammensetzung wird nach 60 Sekunden Inaktivität von Publisher-Teilnehmern in der Stufe automatisch heruntergefahren. An diesem Punkt wird die Zusammensetzung beendet und geht in einen `STOPPED`-Status über. Eine Zusammensetzung wird nach einigen Minuten im `STOPPED`-Status automatisch gelöscht.

# CLI-Anweisungen

Die Verwendung von AWS CLI ist eine Advanced Option und erfordert, dass Sie zuerst die CLI auf Ihrem Computer herunterladen und konfigurieren. Informationen zu den ersten Schritten finden Sie im Benutzerhandbuch für die AWS-Befehlszeilenschnittstelle.

Nun können Sie mit der CLI Ressourcen erstellen und verwalten. Die Zusammensetzungsendpunkte befinden sich unter dem `ivs-realtime` Namespace.

## Erstllen der EncoderConfiguration-Ressource

Eine EncoderConfiguration ist ein Objekt, mit dem Sie das Format des generierten Videos (Höhe, Breite, Bitrate und andere Streaming-Parameter) anpassen können. Sie können eine EncoderConfiguration bei jedem Aufruf des Composition-Endpunkts wiederverwenden, wie im nächsten Schritt erläutert.

Der folgende Befehl erstellt eine EncoderConfiguration-Ressource, die serverseitige Parameter für die Videozusammensetzung wie Videobitrate, Bildrate und Auflösung konfiguriert:

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

Die Antwort lautet:

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

## Starten einer Zusammensetzung

Erstellen Sie mithilfe des in der obigen Antwort bereitgestellten EncoderConfiguration-ARN Ihre Zusammensetzungsressource:

```
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"}}]'
```

Die Antwort zeigt, dass die Zusammensetzung mit einem `STARTING`-Status erstellt wurde. Sobald die Zusammensetzung mit der Veröffentlichung der Zusammensetzung beginnt, geht der Status in `ACTIVE` über. (Sie können den Status anzeigen, indem Sie den Endpunkt ListCompositions oder GetComposition aufrufen.)

Sobald eine Zusammensetzung `ACTIVE` ist, wird die zusammengesetzte Ansicht der IVS-Stufe mithilfe von ListCompositions auf dem IVS-Kanal angezeigt:

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

Die Antwort lautet:

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

Hinweis: Damit die Zusammensetzung aktiv bleibt, müssen die Publisher-Teilnehmer aktiv in der Stufe veröffentlichen. Weitere Informationen finden Sie in den Abschnitten „Veröffentlichen und Abonnieren“ der SDK-Anleitungen für Echtzeit-Streaming-Broadcasts: Web, Android und iOS. Sie müssen für jeden Teilnehmer ein eigenes Stufen-Token erstellen.

## Bildschirmfreigabe aktivieren

Führen Sie die folgenden Schritte aus, um ein festes Bildschirmfreigabe-Layout zu verwenden.

### Erstllen der EncoderConfiguration-Ressource

Mit dem folgenden Befehl wird eine EncoderConfiguration-Ressource erstellt, die serverseitige Zusammensetzungsparameter (Video-Bitrate, Framerate und Auflösung) konfiguriert.

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

Erstellen Sie ein Stufen-Teilnehmer-Token mit einem `screen-share`-Attribut. Da wir `screen-share` als Namen des `featured`-Slots angeben, müssen wir ein Stufen-Token erstellen, bei dem das `screen-share`-Attribut auf `true` festgelegt ist:

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

Die Antwort lautet:

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

## Starten der Zusammensetzung

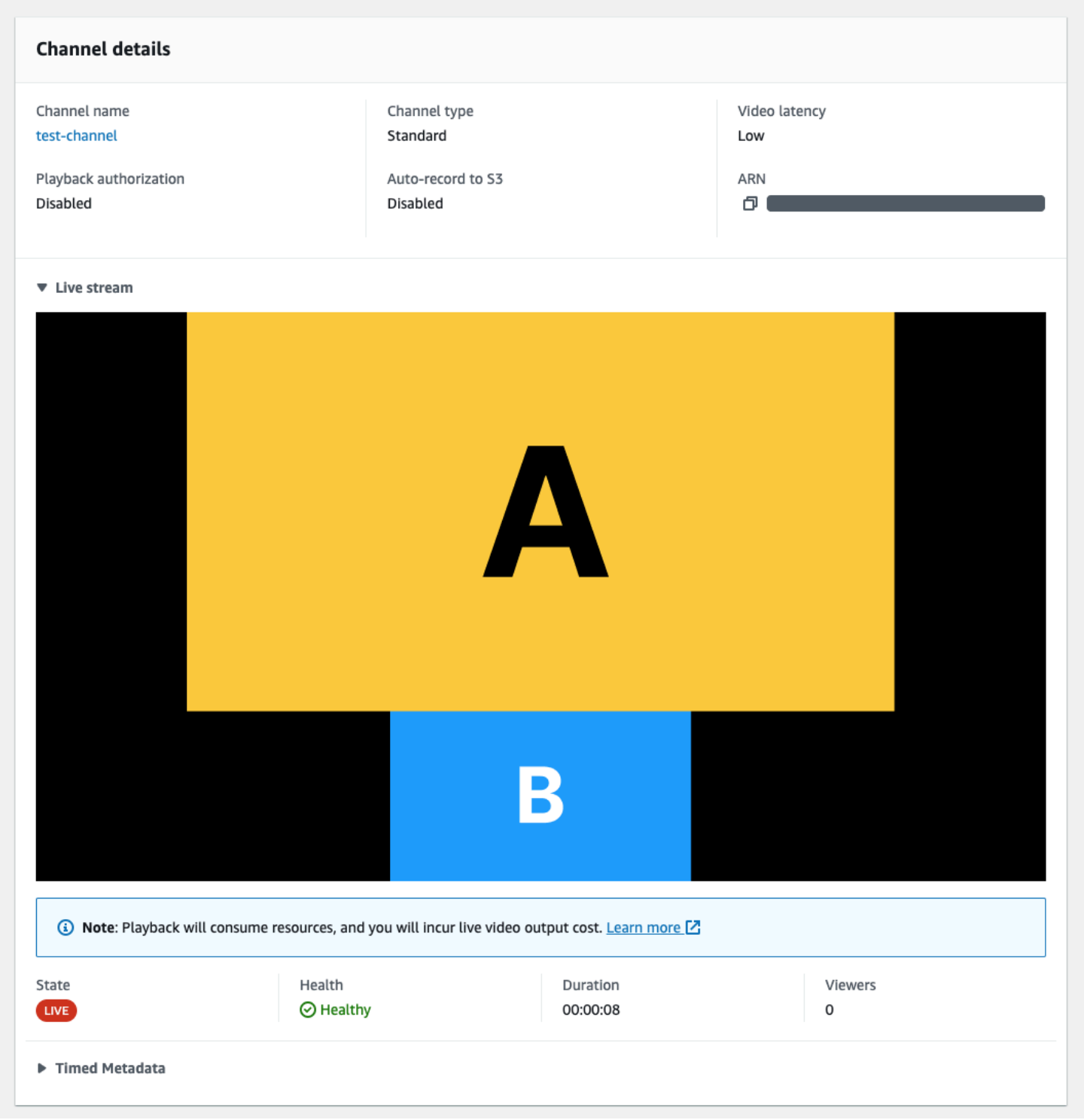
Um die Zusammensetzung mit dem Feature zur Bildschirmfreigabe zu starten, wird dieses Befehl verwendet:

```
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}"
```

Die Antwort lautet:

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

Wenn der Stufen-Teilnehmer `E813MFklPWLF` der Stufe beitritt, wird das Video dieses Teilnehmers im vorgestellten Slot angezeigt und alle anderen Stufen-Publishern werden unterhalb des Slots gerendert:



### Anhalten der Zusammensetzung

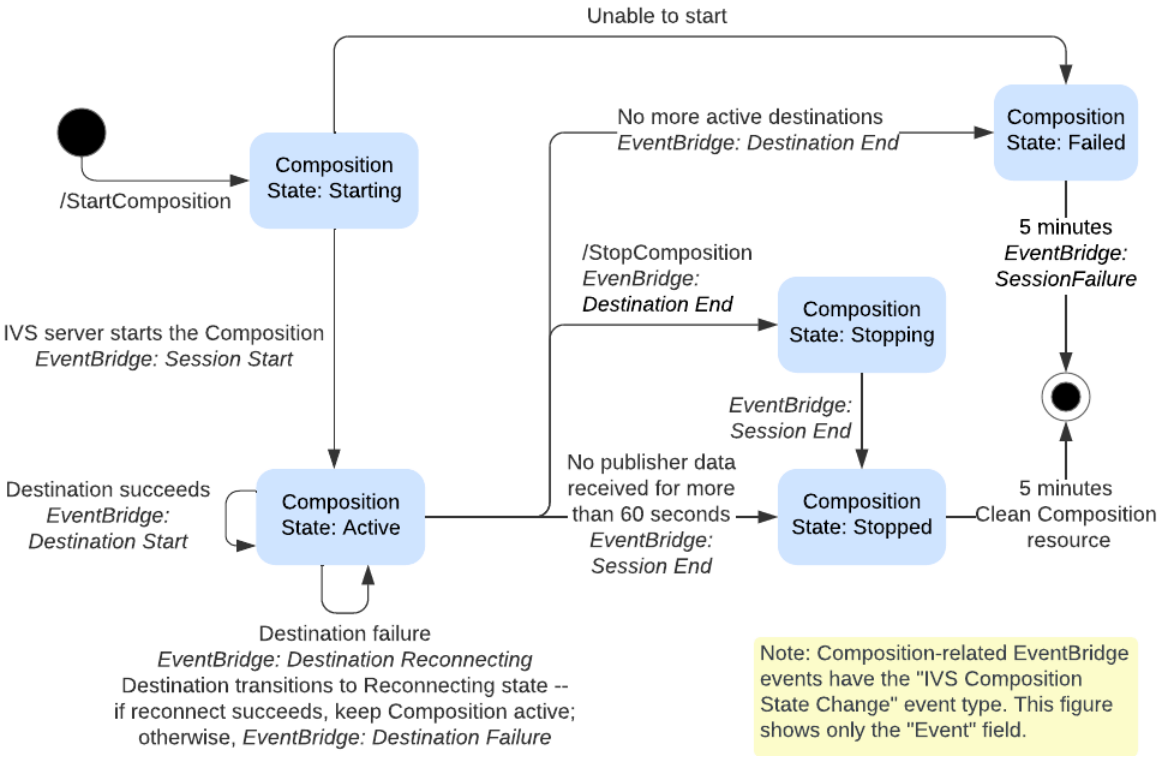
Um eine Zusammensetzung an einem beliebigen Punkt anzuhalten, rufen Sie den StopComposition-Endpunkt auf:

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

## Lebenszyklus einer Zusammensetzung

Verwenden Sie das folgende Diagramm, um sich mit den Statusübergänge einer Zusammensetzung vertraut zu machen. Auf einer hohen Ebene sieht der Lebenszyklus einer Zusammensetzung wie folgt aus:

1. Eine Composition-Ressource wird erstellt, wenn der Benutzer den StartComposition-Endpunkt aufruft
2. Sobald IVS die Zusammensetzung erfolgreich startet, wird ein EventBridge-Ereignis „Statusänderung IVS-Zusammensetzung (Sitzungsstart)“ gesendet. Einzelheiten zu Ereignissen finden Sie unter Verwenden von EventBridge mit IVS-Echtzeit-Streaming.
3. Sobald sich eine Zusammensetzung im aktiven Status befindet, kann Folgendes passieren:
   - Benutzer hält die Zusammensetzung an – Wenn der StopComposition-Endpunkt aufgerufen wird, initiiert IVS ein ordnungsgemäßes Herunterfahren der Zusammensetzung und sendet „Zielende“-Ereignisse, gefolgt von einem „Sitzungsende“-Ereignis.
   - Zusammensetzung wird automatisch herunter gefahren – Wenn kein Teilnehmer in der IVS-Stufe aktiv veröffentlicht, wird die Zusammensetzung nach 60 Sekunden automatisch abgeschlossen und EventBridge-Ereignisse werden gesendet.
   - Zielfehler – Wenn ein Ziel unerwartet ausfällt (z. B. wenn der IVS-Kanal gelöscht wird), wechselt das Ziel in den `RECONNECTING`-Status und es wird ein Ereignis „Wiederverbindung zum Ziel“ gesendet. Wenn eine Wiederherstellung nicht möglich ist, versetzt IVS das Ziel in den `FAILED`-Status und es wird ein „Zielfehler“-Ereignis gesendet. IVS hält die Zusammensetzung aktiv, wenn mindestens eines ihrer Ziele aktiv ist.
4. Sobald sich die Zusammensetzung im `STOPPED`- oder `FAILED`-Status befindet, wird sie nach fünf Minuten automatisch bereinigt. (Dann wird sie nicht mehr von ListCompositions oder GetComposition abgerufen.)


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.

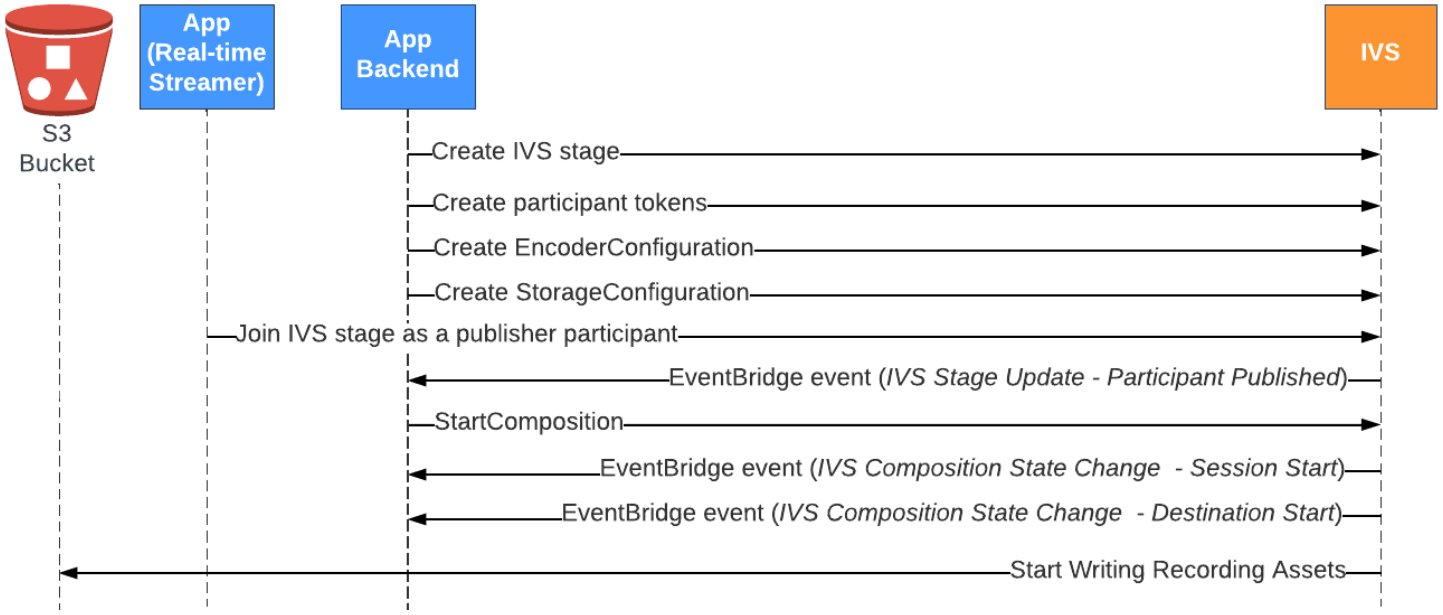
# Zusammengesetzte Aufnahme (Echtzeit-Streaming)

In diesem Dokument wird erläutert, wie Sie das Feature zur Aufzeichnung von Zusammensetzungen innerhalb der serverseitigen Zusammensetzung verwenden. Mit der zusammengesetzten Aufzeichnung können Sie HLS-Aufzeichnungen einer IVS-Stufe generieren, indem Sie mithilfe eines IVS-Servers alle Stufen-Publisher effektiv in einer Ansicht kombinieren und das resultierende Video dann in einem S3-Bucket speichern.

## Voraussetzungen

Um die zusammengesetzte Aufzeichnung verwenden zu können, benötigen Sie eine Stufe mit aktiven Publishern und einen S3-Bucket, der als Aufzeichnungsziel verwendet werden soll. Im Folgenden wird ein möglicher Workflow beschrieben, der EventBridge Ereignisse verwendet, um eine Zusammensetzung in einem S3-Bucket aufzuzeichnen. Alternativ können Sie Zusammensetzungen basierend auf Ihrer eigenen App-Logik starten und stoppen.

1. Erstellen Sie eine IVS-Stufe und Teilnehmer-Token für jeden Publisher.
2. Erstellen Sie ein EncoderConfiguration (ein Objekt, das darstellt, wie das aufgenommene Video gerendert werden soll).
3. Erstellen Sie einen S3-Bucket und einen StorageConfiguration (in dem der Aufzeichnungsinhalt gespeichert wird).
4. Treten Sie der Stufe bei und veröffentlichen Sie dort.
5. Wenn Sie ein vom Teilnehmer veröffentlichtes EventBridge Ereignis erhalten, rufen Sie StartComposition mit einem S3 DestinationConfiguration -Objekt als Ziel auf
6. Nach einigen Sekunden sollten Sie sehen können, dass die HLS-Segmente in Ihren S3-Buckets beibehalten werden.




Hinweis: Eine Zusammensetzung wird nach 60 Sekunden Inaktivität der Publisher-Teilnehmer in der Stufe automatisch heruntergefahren. An diesem Punkt wird die Zusammensetzung beendet und sie geht in einen `STOPPED`-Status über. Eine Zusammensetzung wird nach einigen Minuten im `STOPPED`-Status automatisch gelöscht. Einzelheiten finden Sie unter Zusammensetzungslebenszyklus unter Serverseitige Zusammensetzung.

## Beispiel für eine zusammengesetzte Aufzeichnung: StartComposition mit einem S3-Bucket-Ziel

Das folgende Beispiel zeigt einen typischen Aufruf des StartComposition Endpunkts, wobei S3 als einziges Ziel für die Zusammensetzung angegeben wird. Sobald die Zusammensetzung in einen bestimmten `ACTIVE`-Status übergeht, werden Videosegmente und Metadaten in den durch das `storageConfiguration`-Objekt angegebenen S3-Bucket geschrieben. Informationen zum Erstellen von Zusammensetzungen mit unterschiedlichen Layouts finden Sie unter „Layouts“ im Abschnitt Serverseitige Zusammensetzung und in der API-Referenz zu IVS-Echtzeit-Streaming.

### Anforderung

```
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"
}
```

## Antwort

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

Das in der StartComposition Antwort vorhandene `recordingPrefix` Feld kann verwendet werden, um zu bestimmen, wo der Aufzeichnungsinhalt gespeichert wird.

## Inhalte der Aufnahme

Wenn die Zusammensetzung in einen -`ACTIVE`Zustand übergeht, sehen Sie, dass HLS-Videosegmente und Metadatendateien in den S3-Bucket geschrieben werden, der beim Aufruf von bereitgestellt wurde StartComposition. Diese Inhalte sind für die Nachbearbeitung oder Wiedergabe als On-Demand-Video verfügbar.

Beachten Sie, dass nach dem Liveschalten einer Zusammensetzung das Ereignis „IVS Composition State Change“ ausgegeben wird und es einige Zeit dauern kann, bis die Manifestdateien und Videosegmente geschrieben werden. Es wird empfohlen, aufgezeichnete Streams erst wiederzugeben oder zu verarbeiten, nachdem das Ereignis „IVS Composition State Change (Session End)“ empfangen wurde. Weitere Informationen finden Sie unter Verwenden von EventBridge mit IVS-Echtzeit-Streaming.

Nachfolgend finden Sie eine Beispielverzeichnisstruktur und den Inhalt einer Aufzeichnung einer Live-IVS-Sitzung:

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

Der Ordner `events` enthält die Metadatendateien, die dem Aufzeichnungsereignis entsprechen. JSON-Metadatendateien werden generiert, wenn die Aufzeichnung gestartet, erfolgreich beendet oder mit Fehlern beendet wird:

- `events/recording-started.json`
- `events/recording-ended.json`
- `events/recording-failed.json`

Ein gegebener `events`-Ordner enthält `recording-started.json` und entweder `recording-ended.json` oder `recording-failed.json`.

Diese enthalten Metadaten, die sich auf die aufgezeichnete Sitzung und ihre Ausgabeformate beziehen. JSON-Details sind unten angegeben.

Der Ordner `media` enthält die unterstützten Medieninhalte. Der Unterordner `hls` enthält alle Medien und Manifestdateien, die während der Zusammensetzungssitzung generiert wurden, und kann mit dem IVS-Player abgespielt werden. Das HLS-Manifest befindet sich im Ordner `multivariant.m3u8`.

## Bucket-Richtlinie für StorageConfiguration

Wenn ein StorageConfiguration Objekt erstellt wird, erhält IVS Zugriff, um Inhalte in den angegebenen S3-Bucket zu schreiben. Dieser Zugriff wird durch Änderungen an der Richtlinie des S3-Buckets gewährt. Wenn die Richtlinie für den Bucket so geändert wird, dass der Zugriff von IVS aufgehoben wird, schlagen laufende und neue Aufzeichnungen fehl.

Das folgende Beispiel zeigt eine S3-Bucket-Richtlinie, die IVS das Schreiben in den S3-Bucket ermöglicht:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CompositeWrite-y1d212y",
            "Effect": "Allow",
            "Principal": {
                "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
            },
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::my-s3-bucket/*",
            "Condition": {
```

```
                "StringEquals": {
                    "s3:x-amz-acl": "bucket-owner-full-control"
                },
                "Bool": {
                    "aws:SecureTransport": "true"
                }
            }
        }
    ]
}
```

## JSON-Metadatendateien

Diese Metadaten weisen das JSON-Format auf. Es enthält die folgenden Informationen:

| Feld | Typ | Erforderlich | Beschreibung |
|---|---|---|---|
| `stage_arn` | Zeichenfolge | Ja | ARN der Stufe, die als Quelle für die Zusammensetzung verwendet wird. |
| `media` | object | Ja | Objekt, das die Aufzählungsobjekte von Medieninhalten enthält, die für diese Aufzeichnung verfügbar sind. Zulässige Werte: `"hls"`. |
| `hls` | object | Ja | Aufzählungsfeld, das die Ausgabe des Apple HLS-Formats beschreibt. |
| `duration_ms` | Ganzzahl | Bedingt | Dauer des aufgezeichneten HLS-Inhalts in Millisekunden. Dies ist nur verfügbar, wenn `recording_status` `"RECORDING_ENDED"` oder `"RECORDING_ENDED_WITH_FAILURE"` ist. Wenn ein Fehler aufgetret |

| Feld | Typ | Erforderlich | Beschreibung |
|---|---|---|---|
| | | | en ist, bevor eine Aufzeichnung durchgeführt wurde, ist dies 0. |
| `path` | Zeichenfolge | Ja | Relativer Pfad vom S3-Präfix, in dem HLS-Inhalt gespeichert wird. |
| `playlist` | Zeichenfolge | Ja | Name der HLS-Master-Wiedergabeliste. |
| `renditions` | Objekt | Ja | Array von Formatvarianten (HLS-Variante) von Metadatenobjekten. Es ist immer mindestens eine Formatvariante vorhanden. |
| `path` | Zeichenfolge | Ja | Relativer Pfad vom S3-Präfix, in dem HLS-Inhalt für diese Formatvariante gespeichert wird. |
| `playlist` | Zeichenfolge | Ja | Name der Medienwiedergabeliste für diese Formatvariante. |
| `resolution_height` | int | Bedingt | Pixelauflösungshöhe des codierten Videos. Diese Option ist nur verfügbar, wenn die Formatvariante eine Videospur enthält. |
| `resolution_width` | int | Bedingt | Pixelauflösungsbreite des codierten Videos. Diese Option ist nur verfügbar, wenn die Formatvariante eine Videospur enthält. |

| Feld | Typ | Erforderlich | Beschreibung |
|---|---|---|---|
| `recording_ended_at` | Zeichenfolge | Bedingt | RFC 3339 UTC-Zeitstempel, wenn die Aufnahme beendet wurde. Dies ist nur verfügbar, wenn `recording_status` `"RECORDING_ENDED"` oder `"RECORDING_ENDED_WITH_FAILURE"` ist.<br><br>`recording_started_at` und `recording_ended_at` sind Zeitstempel, wenn diese Ereignisse generiert werden, und stimmen möglicherweise nicht genau mit den Zeitstempeln des HLS-Videosegments überein. Um die Dauer einer Aufnahme genau zu bestimmen, verwenden Sie das Feld `duration_ms`. |
| `recording_started_at` | Zeichenfolge | Bedingt | RFC 3339 UTC-Zeitstempel, wenn die Aufnahme gestartet wurde. Dies ist nicht verfügbar, wenn sich der `recording_status` in `RECORDING_START_FAILED` befindet.<br><br>Beachten Sie den oben stehenden Hinweis zu `recording_ended_at`. |

| Feld | Typ | Erforderlich | Beschreibung |
|---|---|---|---|
| `recording_status` | Zeichenfolge | Ja | Aufzeichnungsstatus. Zulässige Werte: `"RECORDING_STARTED"`, `"RECORDING_ENDED"`, `"RECORDING_START_FAILED"`, `"RECORDING_ENDED_WITH_FAILURE"`. |
| `recording_status_message` | Zeichenfolge | Bedingt | Beschreibende Informationen über den Status. Dies ist nur verfügbar, wenn `recording_status` `"RECORDING_ENDED"` oder `"RECORDING_ENDED_WITH_FAILURE"` ist. |
| `version` | Zeichenfolge | Ja | Die Version des Metadatenschemas. |

## Beispiel: 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
        }
      ]
```

```
    }
  }
}
```

## Beispiel: 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
        }
      ]
    }
  }
}
```

## Beispiel: 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
        }
      ]
    }
  }
}
```

# Wiedergabe von aufgezeichneten Inhalten aus privaten Buckets

Standardmäßig sind die aufgezeichneten Inhalte privat. Aus diesem Grund ist die Wiedergabe dieser Objekte über die direkte S3-URL nicht möglich. Wenn Sie versuchen, die multivariate HLS-Wiedergabeliste (m3u8-Datei) zur Wiedergabe mit dem IVS-Player oder einem anderen Player zu öffnen, erhalten Sie eine Fehlermeldung (z. B. „Sie haben keine Berechtigung zum Zugriff auf die angeforderte Ressource“). Stattdessen können Sie diese Dateien mit dem Amazon CloudFront CDN (Content Delivery Network) wiedergeben.

CloudFront -Verteilungen können so konfiguriert werden, dass Inhalte aus privaten Buckets bereitgestellt werden. In der Regel ist dies gegenüber öffentlich zugänglichen Buckets vorzuziehen, in denen Lesevorgänge die von angebotenen Kontrollen umgehen CloudFront. Sie können Ihre Verteilung so einrichten, dass sie von einem privaten Bucket bereitgestellt wird, indem Sie eine Ursprungszugriffssteuerung (OAC) erstellen. Dabei handelt es sich um einen speziellen CloudFront Benutzer, der über Leseberechtigungen für den privaten Ursprungs-Bucket verfügt. Sie können die OAC erstellen, nachdem Sie Ihre Verteilung erstellt haben, über die CloudFront Konsole oder API. Weitere Informationen finden Sie unter Erstellen einer neuen Ursprungszugriffssteuerung im Amazon-CloudFront Entwicklerhandbuch.

## Einrichten der Wiedergabe mithilfe von CloudFront mit aktiviertem CORS

In diesem Beispiel wird beschrieben, wie ein Entwickler eine CloudFront Verteilung mit aktiviertem CORS einrichten und die Wiedergabe seiner Aufzeichnungen von jeder Domain aus ermöglichen kann. Dies ist besonders während der Entwicklungsphase nützlich. Sie können das folgende Beispiel jedoch an Ihre Produktionsanforderungen anpassen.

## Schritt 1: S3-Bucket erstellen

Erstellen Sie einen S3-Bucket, der zum Speichern der Aufzeichnungen verwendet wird. Beachten Sie, dass sich der Bucket in derselben Region befinden muss, die Sie für Ihren IVS-Workflow verwenden.

Fügen Sie dem Bucket eine CORS-Berechtigungserichtlinie hinzu:

1. Navigieren Sie in der AWS-Konsole zur Registerkarte S3-Bucket-Berechtigungen.
2. Kopieren Sie die untenstehende CORS-Richtlinie und fügen Sie sie unter Cross-Origin Resource Sharing (CORS) ein. Dadurch wird der CORS-Zugriff auf den S3-Bucket aktiviert.

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

## Schritt 2: Erstellen einer CloudFront Verteilung

Weitere Informationen finden Sie unter Erstellen einer CloudFront Verteilung im CloudFront - Entwicklerhandbuch.

Geben Sie über die AWS-Konsole die folgenden Informationen ein:

| Für dieses Feld … | Wählen Sie dies ... |
| --- | --- |
| Ursprungs-Domain | Der im vorherigen Schritt erstellte S3-Bucket |
| Ursprungszugriff | Einstellungen für die Ursprungs-Zugriffs kontrolle (empfohlen) unter Verwendung von Standardparametern |
| Standard-Cache-Verhalten: Viewer-Protokollrichtlinie | Redirect HTTP to HTTPS |
| Standard-Cache-Verhalten: Zulässige HTTP-Methoden | GET, HEAD und OPTIONS |
| Standard-Cache-Verhalten: Cache-Schlüssel- und Ursprungsanfragen | CachingDisabled Richtlinie |
| Standard-Cache-Verhalten: Ursprungs-Anforderungsrichtlinie | CORS-S3Origin |
| Standard-Cache-Verhalten: Richtlinie für Antwort-Header | SimpleCORS |
| Webanwendungs-Firewall | Aktivieren von Sicherheitsmaßnahmen |

Speichern Sie dann die CloudFront Verteilung.

## Schritt 3: Einrichten der S3-Bucket-Richtlinie

1. Löschen Sie alle StorageConfiguration , die Sie für den S3-Bucket eingerichtet haben. Dadurch werden alle Bucket-Richtlinien entfernt, die beim Erstellen der Richtlinie für diesen Bucket automatisch hinzugefügt wurden.
2. Gehen Sie zu Ihrer CloudFront Verteilung, stellen Sie sicher, dass sich alle Verteilungsfelder in den im vorherigen Schritt definierten Zuständen befinden, und kopieren Sie die Bucket-Richtlinie (verwenden Sie die Schaltfläche Richtlinie kopieren).
3. Navigieren Sie zu Ihrem S3-Bucket. Wählen Sie auf der Registerkarte Berechtigungen die Option Bucket-Richtlinie bearbeiten aus und fügen Sie die Bucket-Richtlinie ein, die Sie im vorherigen

Schritt kopiert haben. Nach diesem Schritt sollte die Bucket-Richtlinie ausschließlich über die CloudFront Richtlinie verfügen.

4. Erstellen Sie eine und StorageConfigurationgeben Sie den S3-Bucket an.

Nachdem die erstellt StorageConfiguration wurde, sehen Sie zwei Elemente in der S3-Bucket-Richtlinie, eines ermöglicht CloudFront das Lesen von Inhalten und eines, das IVS das Schreiben von Inhalten ermöglicht. Ein Beispiel für eine endgültige Bucket-Richtlinie mit - CloudFront und IVS-Zugriff finden Sie unter Beispiel: S3-Bucket-Richtlinie mit CloudFront und IVS-Zugriff.

### Schritt 4: Wiedergabe von Aufnahmen

Nachdem Sie die CloudFront Verteilung erfolgreich eingerichtet und die Bucket-Richtlinie aktualisiert haben, sollten Sie in der Lage sein, Aufzeichnungen mit dem IVS-Player wiederzugeben:

1. Starten Sie erfolgreich eine Zusammensetzung und stellen Sie sicher, dass eine Aufzeichnung im S3-Bucket gespeichert ist.
2. Nachdem Sie Schritt 1 bis Schritt 3 in diesem Beispiel befolgt haben, sollten die Videodateien über die CloudFront URL zur Verwendung verfügbar sein. Ihre CloudFront URL ist der Name der Verteilungsdomäne auf der Registerkarte Details in der Amazon- CloudFront Konsole. Sie ist in etwa wie folgt:

   ```
   a1b23cdef4ghij.cloudfront.net
   ```

3. Um das aufgenommene Video über die CloudFront Verteilung abzuspielen, suchen Sie den Objektschlüssel für Ihre `multivariant.m3u8` Datei im S3-Bucket. Sie ist in etwa wie folgt:

   ```
   FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/
   multivariant.m3u8
   ```

4. Hängen Sie den Objektschlüssel an das Ende Ihrer CloudFront URL an. Ihre endgültige URL sieht etwa folgendermaßen aus:

   ```
   https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/
   fjFKbylPb3k4/composite/media/hls/multivariant.m3u8
   ```

5. Sie können jetzt die endgültige URL zum Quellattribut eines IVS-Players hinzufügen, um die vollständige Aufzeichnung anzusehen. Um das aufgezeichnete Video anzusehen, können Sie die Demo unter Erste Schritte im IVS Player SDK: Web Guide verwenden.

# Beispiel: S3-Bucket-Richtlinie mit CloudFront und IVS-Zugriff

Der folgende Codeausschnitt veranschaulicht eine S3-Bucket-Richtlinie, die es ermöglicht, Inhalte CloudFront in den privaten Bucket zu lesen und IVS Inhalte in den Bucket zu schreiben. Hinweis: Kopieren Sie den unten stehenden Ausschnitt nicht und fügen Sie ihn nicht in Ihren eigenen Bucket ein. Ihre Richtlinie sollte die IDs enthalten, die für Ihre CloudFront Verteilung und relevant sind 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"
```

```
        }
      }
    }
  ]
}
```

## Fehlerbehebung

- Die Zusammensetzung wird nicht in den S3-Bucket geschrieben – Stellen Sie sicher, dass der S3-Bucket und die StorageConfiguration Objekte erstellt werden und sich in derselben Region befinden. Stellen Sie außerdem sicher, dass IVS Zugriff auf den Bucket hat, indem Sie Ihre Bucket-Richtlinie überprüfen; siehe Bucket-Richtlinie für StorageConfiguration.
- Ich kann bei der Ausführung keine Zusammensetzung finden ListCompositions – Zusammensetzungen sind kurzlebige Ressourcen. Sobald diese in einen endgültigen Status übergehen, werden sie nach einigen Minuten automatisch gelöscht.
- Meine Zusammensetzung hält automatisch an – Eine Zusammensetzung stoppt automatisch, wenn sich mehr als 60 Sekunden lang kein Publisher in der Stufe befindet.

## Bekanntes Problem

Die von der zusammengesetzten Aufhzeichnung geschriebene Medien-Wiedergabeliste erhält das Tag `#EXT-X-PLAYLIST-TYPE:EVENT`, während die Zusammensetzung läuft. Wenn die Zusammensetzung abgeschlossen ist, wird das Tag auf `#EXT-X-PLAYLIST-TYPE:VOD` aktualisiert. Für ein reibungsloses Wiedergabeerlebnis empfiehlt es sich, diese Wiedergabeliste erst zu verwenden, nachdem die Zusammensetzung erfolgreich abgeschlossen wurde.

# OBS- und WHIP-Unterstützung (Echtzeit-Streaming)

In diesem Dokument wird erläutert, wie Sie WHIP-kompatible Encoder wie OBS verwenden, um in IVS-Echtzeit-Streaming zu veröffentlichen. WHIP (WebRTC -HTTP Ingestion Protocol) ist ein IETF-Entwurf, der zur Standardisierung der WebRTC-Erfassung entwickelt wurde.

WHIP ermöglicht die Kompatibilität mit Software wie OBS und bietet eine Alternative (zum IVS-Broadcast-SDK) für die Desktop-Veröffentlichung. Erfahrenere Streamer, die mit OBS vertraut sind, bevorzugen es möglicherweise aufgrund seiner erweiterten Produktionsfeatures wie Szenenübergänge, Audiomischung und Overlay-Grafiken. Dies bietet Entwicklern eine hervorragende Option: Verwenden Sie das IVS-Web-Broadcast-SDK für die direkte Browserveröffentlichung oder erlauben Sie Streamern, OBS auf ihrem Desktop zu verwenden, um leistungsfähigere Tools zu erhalten.

Außerdem ist WHIP in Situationen von Vorteil, in denen die Verwendung des IVS-Broadcast-SDK nicht möglich oder bevorzugt ist. Beispielsweise ist das IVS-Broadcast-SDK in Setups mit Hardware-Encodern möglicherweise keine Option. Wenn der Encoder jedoch WHIP unterstützt, können Sie trotzdem direkt vom Encoder in IVS veröffentlichen.

## OBS-Handbuch

OBS unterstützt WHIP ab Version 30. Laden Sie zunächst OBS v30 oder höher herunter: https://obsproject.com/.

Gehen Sie folgendermaßen vor, um mit OBS über WHIP in einer IVS-Stufe zu veröffentlichen:

1. Generieren Sie ein Teilnehmer-Token mit Veröffentlichungsfunktion. In WHIP-Begriffen ist ein Teilnehmer-Token ein Bearer-Token. Standardmäßig laufen Teilnehmer-Token in 12 Stunden ab, aber Sie können die Dauer auf bis zu 14 Tage verlängern.
2. Klicken Sie auf Settings (Einstellungen). Wählen Sie im Bereich Stream des Bereichs Einstellungen die Option WHIP aus der Dropdownliste Service aus.
3. Geben Sie für den Server https://global.whip.live-video.net/ ein.
4. Geben Sie für das Bearer-Token das Teilnehmer-Token ein, das Sie in Schritt 2 generiert haben.
5. Konfigurieren Sie Ihre Videoeinstellungen wie gewohnt, mit einigen Einschränkungen:
   a. IVS-Echtzeit-Streaming unterstützt Eingaben bis 720p bei 8,5 Mbit/s. Wenn Sie einen dieser Grenzwerte überschreiten, wird Ihr Stream getrennt.

b. Wir empfehlen, Ihr Keyframe-Intervall im Ausgabebereich auf 1s oder 2s festzulegen. Ein niedriges Keyframe-Intervall ermöglicht es Zuschauern, die Videowiedergabe schneller zu starten. Wir empfehlen außerdem, die CPU-Nutzungsvoreinstellung auf ultraschnell und auf Nulllatenz einzustellen, um die niedrigste Latenz zu ermöglichen.

c. Da OBS Simulcast nicht unterstützt, empfehlen wir, Ihre Bitrate unter 2,5 Mbit/s zu halten. Auf diese Weise können Betrachter Verbindungen mit geringerer Bandbreite beobachten.

6. Drücken Sie Streaming starten .

# Service Quotas (Echtzeit-Streaming)

Im Folgenden finden Sie Service Quotas und Limits für Amazon Interactive Video Service (IVS)-Endpunkte, Ressourcen und andere Vorgänge. Service Quotas (auch als Limits bezeichnet) sind die maximale Anzahl von Service-Ressourcen oder Vorgängen für Ihr AWS-Konto. Das heißt, diese Grenzwerte gelten je AWS Konto, sofern in der Tabelle nichts anderes angegeben ist. Lesen Sie auch den Abschnitt AWS-Service-Quotas.

Um programmgesteuert eine Verbindung zu einem AWS-Service herzustellen, verwenden Sie einen Endpunkt. Lesen Sie auch den Abschnitt AWS-Service-Endpunkte.

Alle Kontingente werden pro Region erzwungen.

## Erhöhte Service Quotas

Für einstellbare Kontingente können Sie eine Ratenerhöhung über die AWS-Konsole anfragen. Verwenden Sie die Konsole, um Informationen über Service Quotas anzuzeigen.

Kontingente für API-Anrufraten sind nicht anpassbar.

## API-Aufrufratenquoten

| Endpunkttyp | Endpunkt | Standard |
|---|---|---|
| Composition | `GetComposition` | 5 TPS |
| Composition | `ListCompositions` | 5 TPS |
| Composition | `StartComposition` | 5 TPS |
| Composition | `StopComposition` | 5 TPS |
| MediaEncoder | `CreateEncoderConfiguration` | 5 TPS |
| MediaEncoder | `DeleteEncoderConfiguration` | 5 TPS |
| MediaEncoder | `GetEncoderConfiguration` | 5 TPS |
| MediaEncoder | `ListEncoderConfigurations` | 5 TPS |

| Endpunkttyp | Endpunkt | Standard |
|---|---|---|
| Stufe | `CreateParticipantToken` | 50 TPS |
| Stufe | `CreateStage` | 5 TPS |
| Stufe | `DeleteStage` | 5 TPS |
| Stufe | `DisconnectParticipant` | 5 TPS |
| Stufe | `GetParticipant` | 5 TPS |
| Stufe | `GetStage` | 5 TPS |
| Stufe | `GetStageSession` | 5 TPS |
| Stufe | `ListStages` | 5 TPS |
| Stufe | `UpdateStage` | 5 TPS |
| Stufe | `ListParticipants` | 5 TPS |
| Stufe | `ListParticipantEvents` | 5 TPS |
| Stufe | `ListStageSessions` | 5 TPS |
| StorageConfiguration | `CreateStorageConfiguration` | 5 TPS |
| StorageConfiguration | `DeleteStorageConfiguration` | 5 TPS |
| StorageConfiguration | `GetStorageConfiguration` | 5 TPS |
| StorageConfiguration | `ListStorageConfigurations` | 5 TPS |
| Tags | `ListTagsForResource` | 10 TPS |
| Tags | `TagResource` | 10 TPS |
| Tags | `UntagResource` | 10 TPS |

# Andere Kontingente

| Ressource oder Feature | Standard | Anpassbar | Beschreibung |
|---|---|---|---|
| EncoderConfigurations | 20 | Ja | Maximale Anzahl der Encoder-Konfigurationsressourcen pro Konto. |
| Ziele für die Zusammensetzung | 2 | Nein | Maximale Anzahl von Zielobjekten in einer Zusammensetzungsressource. |
| Zusammensetzung: maximale Dauer | 24 | Nein | Maximale Dauer, für die eine Zusammensetzung existieren kann, in Stunden. |
| Zusammensetzungen | 5 | Ja | Maximale Anzahl gleichzeitiger Zusammensetzungsressourcen pro Konto. |
| Dauer der Veröffentlichung oder des Abonnements eines Teilnehmers | 24 | Nein | Die maximale Zeitspanne (in Stunden), in der ein Teilnehmer etwas auf einer Bühne veröffentlichen oder diese abonnieren kann. |
| Teilnehmer veröffentlicht Resolution | 720p | Nein | Maximale Auflösung des von den Teilnehmern veröffentlichten Videos. |
| Download-Bitrate des Teilnehmers | 8,5 Mbit/s | Nein | Maximale aggregierte Download-Bitrate für alle Abonnements eines Teilnehmers. |
| Teilnehmer der Bühne (Publisher) | 12 | Nein | Höchstzahl von Teilnehmern, die gleichzeitig zu einer Bühne veröffentlichen können. |

| Ressource oder Feature | Standard | Anpassbar | Beschreibung |
| --- | --- | --- | --- |
| Teilnehmer der Bühne (Subscriber) | 10.000 | Ja | Höchstzahl von Teilnehmern, die gleichzeitig eine Bühne abonnieren können. |
| Phasen | 100 | Ja | Höchstzahl von Stages pro AWS-Region |

# Streaming-Optimierungen in Echtzeit

Um sicherzustellen, dass Ihre Benutzer das beste Erlebnis beim Streamen und Ansehen von Videos mithilfe von IVS Echtzeit-Streaming haben, gibt es verschiedene Möglichkeiten, das Erlebnis zu verbessern oder Teile des Erlebnisses zu optimieren, indem Sie die Features verwenden, die wir heute anbieten.

## Einführung

Bei der Optimierung der Benutzererlebnisqualität ist es wichtig, die gewünschte Benutzererfahrung zu berücksichtigen. Diese kann sich je nach den Inhalten, die sie sich ansehen, und den Netzwerkbedingungen ändern.

In diesem Leitfaden konzentrieren wir uns auf Benutzer, die entweder Publisher von Streams oder Subscriber von Streams sind und wir berücksichtigen die gewünschten Aktionen und Erfahrungen dieser Nutzer.

## Adaptives Streaming: Mehrschichtige Kodierung mit Simulcast

Dieses Feature wird nur von den folgenden Client-Versionen unterstützt:

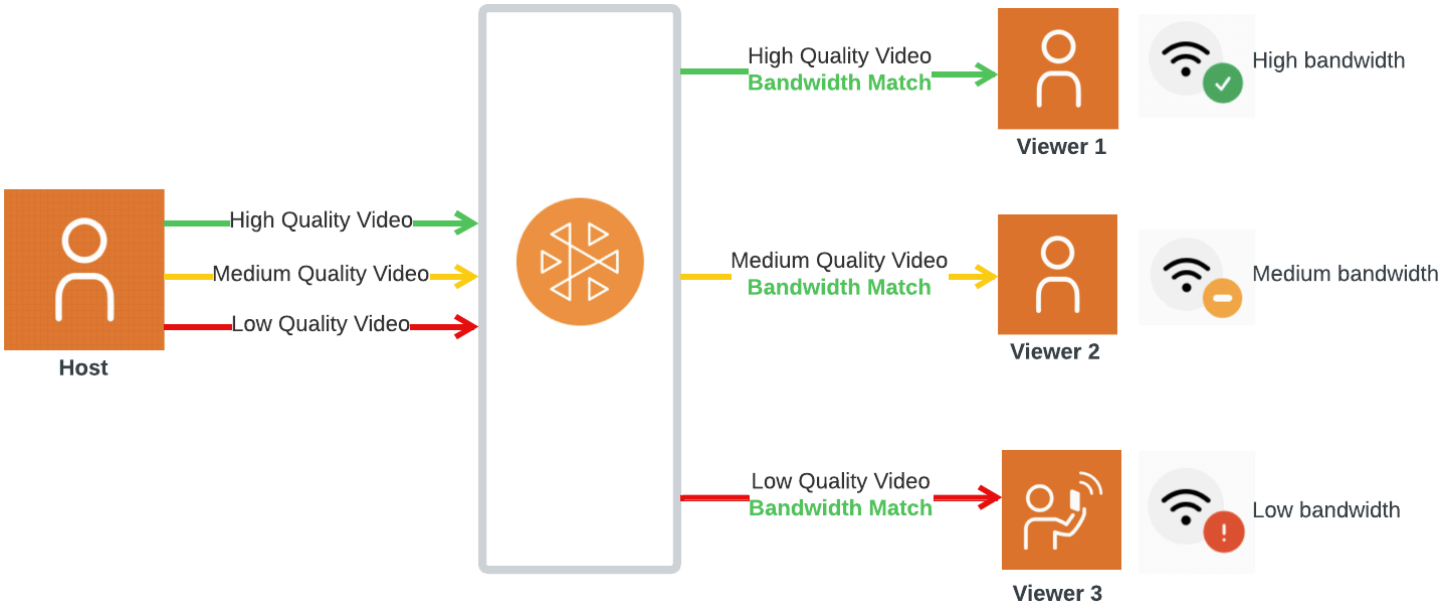
- iOS und Android 1.12.0+
- Web 1.5.1+

Sie müssen eine E-Mail an amazon-ivs-simulcast@amazon.com senden, um sich für dieses Feature für Ihr Konto anzumelden. Die Aktivierung von Simulcast über die SDK-Konfiguration hat keine Auswirkungen, sofern Sie nicht angemeldet sind.

Sobald Sie sich für dieses Feature entschieden haben, codieren die Publisher bei Verwendung der IVS-SDKs für Echtzeitübertragungen mehrere Ebenen von Videos, und die Subscriber passen sich automatisch an die für ihr Netzwerk am besten geeignete Qualität an bzw. wechseln zu dieser. Wir nennen das mehrschichtige Kodierung mit Simulcast.

Die mehrschichtige Kodierung mit Simulcast wird auf Android und iOS sowie auf Chrome-Desktopbrowsern (für Windows und macOS) unterstützt. Wir unterstützen keine mehrschichtige Kodierung in anderen Browsern.

In der Abbildung unten sendet der Host drei Videoqualitäten (hoch, mittel und niedrig). IVS leitet auf der Grundlage der verfügbaren Bandbreite das Video von höchster Qualität an jeden Zuschauer weiter. Dies bietet jedem Zuschauer ein optimales Erlebnis. Wenn sich die Netzwerkverbindung von Zuschauer 1 von gut zu schlecht ändert, beginnt IVS automatisch damit, Zuschauer 1 Video mit geringerer Qualität zu senden, sodass Zuschauer 1 den Stream weiterhin ohne Unterbrechung ansehen kann (mit der bestmöglichen Qualität).




## Standardebenen, Qualitäten und Frameraten

Die Standardqualitäten und Ebenen, die für Mobil- und Webbenutzer bereitgestellt werden, lauten wie folgt:

| Mobil (Android, iOS) | Web (Chrome) |
| --- | --- |
| Hohe Schicht (oder benutzerdefiniert):<br>• Maximale Bitrate: 900 000 bps<br>• Bildrate: 15 fps<br>• Auflösung: 360x640 | Hohe Schicht (oder benutzerdefiniert):<br>• Maximale Bitrate: 1 700 000 bps<br>• Bildrate: 30 fps<br>• Auflösung: 1280x720 |
| Mittelschicht: keine (nicht erforderlich, da der Unterschied zwischen | Mittelschicht:<br>• Maximale Bitrate: 700 000 bps |

| Mobil (Android, iOS) | Web (Chrome) |
| --- | --- |
| den Bitraten auf hoher und niedriger Ebene auf Mobilgeräten gering ist) | • Bildrate: 20 fps<br>• Auflösung: 640x360 |
| Niedrige Schicht:<br>• Maximale Bitrate: 150 000 bps<br>• Bildrate: 15 fps<br>• Auflösung: 180x320 | Niedrige Schicht:<br>• Maximale Bitrate: 200 000 bps<br>• Bildrate: 15 fps<br>• Auflösung: 320x180 |

## Konfiguration von mehrschichtiger Kodierung mit Simulcast

Um die mehrschichtige Kodierung mit Simulcast verwenden zu können, müssen Sie sich für das Feature entschieden und dies auf dem Client aktiviert haben. Wenn Sie es aktivieren, wird eine Erhöhung der Gesamtbitrate übertragen, mit dem Vorteil, dass weniger Videos eingefroren werden.

Android

```
// Opt-out of Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
// Opt-out of Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

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

// Other Stage implementation code
```

Web

```
// Opt-out of Simulcast
```

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

// Other Stage implementation code
```

# Streamingkonfigurationen

In diesem Abschnitt werden andere Konfigurationen beschrieben, die Sie an Ihren Video- und Audiostreams vornehmen können.

## Ändern der Bitrate des Video-Streams

Verwenden Sie die folgenden Konfigurationsbeispiele, um die Bitrate Ihres Video-Streams zu ändern.

Android

```
StageVideoConfiguration config = new StageVideoConfiguration();

// Update Max Bitrate to 1.5mbps
config.setMaxBitrate(1500000);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamVideoConfiguration();

// Update Max Bitrate to 1.5mbps
try! config.setMaxBitrate(1500000);

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

// Other Stage implementation code
```

Web

```
// Note: On web it is also recommended to configure the framerate of your device from
 userMedia
```

```
const camera = await navigator.mediaDevices.getUserMedia({
   video: {
      bitrate: {
         ideal: 1500,
         max: 1500,
      },
   },
});

let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Bitrate to 1.5mbps or 1500kbps
   maxBitrate: 1500
})

// Other Stage implementation code
```

## Ändern der Framerate des Video-Streams

Verwenden Sie die folgenden Konfigurationsbeispiele, um die Framerate Ihres Video-Streams zu ändern.

### Android

```
StageVideoConfiguration config = new StageVideoConfiguration();

// Update target framerate to 10fps
config.targetFramerate(10);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

### iOS

```
let config = IVSLocalStageStreamVideoConfiguration();

// Update target framerate to 10fps
try! config.targetFramerate(10);

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

// Other Stage implementation code
```

Web

```
// Note: On web it is also recommended to configure the framerate of your device from
 userMedia
const camera = await navigator.mediaDevices.getUserMedia({
   video: {
      frameRate: {
         ideal: 10,
         max: 10,
      },
   },
});

let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Framerate to 10fps
   maxFramerate: 10
})
// Other Stage implementation code
```

## Optimieren der Audio-Bitrate und Stereo-Unterstützung

Verwenden Sie die folgenden Konfigurationsbeispiele, um die Bitrate und Stereoeinstellungen Ihres Audio-Streams zu ändern.

Web

```
// Note: Disable autoGainControl, echoCancellation, and noiseSuppression when enabling
 stereo.
const camera = await navigator.mediaDevices.getUserMedia({
   audio: {
      autoGainControl: false,
      echoCancellation: false,
      noiseSuppression: false
   },
});

let audioStream = new LocalStageStream(camera.getAudioTracks()[0], {
   // Optional: Update Max Audio Bitrate to 96Kbps. Default is 64Kbps
   maxAudioBitrateKbps: 96,

   // Signal stereo support. Note requires dual channel input source.
   stereo: true
```

```
})

// Other Stage implementation code
```

Android

```
StageAudioConfiguration config = new StageAudioConfiguration();

// Update Max Bitrate to 96Kbps. Default is 64Kbps.
config.setMaxBitrate(96000);

AudioLocalStageStream microphoneStream = new AudioLocalStageStream(microphone, config);

// Other Stage implementation code
```

iOS

```
let config = IVSLocalStageStreamConfiguration();

// Update Max Bitrate to 96Kbps. Default is 64Kbps.
try! config.audio.setMaxBitrate(96000);

let microphoneStream = IVSLocalStageStream(device: microphone, config: config);

// Other Stage implementation code
```

# Empfohlene Optimierungen

| Szenario | Empfehlungen |
| --- | --- |
| Streams mit Text oder sich langsam bewegenden Inhalten wie Präsentat ionen oder Folien | Verwenden Sie für Simulcast die Codierung auf mehreren Ebenen oder konfigurieren Sie Streams mit einer niedrigeren Framerate. |
| Streams mit Action oder viel Bewegung | Verwenden Sie für Simulcast die Codierung auf mehreren Ebenen. |
| Ströme mit Konversation oder wenig Bewegung | Verwenden Sie für Simulcast die Codierung auf mehreren Ebenen oder wählen Sie „Nur Audio“ (siehe „Teilnehmer |

| Szenario | Empfehlungen |
| --- | --- |
| | abonnieren“ in den SDK-Leitfäden für Echtzeit-Streaming-Broadcasts: Web, Android und iOS.) |
| Nutzer streamen mit begrenztem Datenvolumen | Verwenden Sie für Simulcast die Codierung auf mehreren Ebenen oder, wenn Sie eine geringere Datennutzung für alle wünschen, konfigurieren Sie eine niedrigere Framerate und senken Sie die Bitrate manuell. |

# Ressourcen und Unterstützung (Echtzeit-Streaming)

## Ressourcen

https://ivs.rocks/ ist eine spezielle Website zum Durchsuchen veröffentlichter Inhalte (Demos, Codebeispiele, Blog-Posts), Kostenschätzungen und Erleben von Amazon IVS durch Live-Demos.

## Demos


Die IVS-Echtzeit-Streaming-Demo für iOS und Android zeigt Entwicklern, wie sie Amazon IVS verwenden können, um eine überzeugende, von sozialen Benutzern generierte, Echtzeitanwendung für Inhalte zu erstellen. Diese Anwendung bietet einen scrollbaren Feed mit benutzergenerierten Echtzeit-Streams. Benutzer können Videostreams und Nur-Audioräume erstellen. Videostream-Gäste können im Gastmodus oder im Versus-Modus (VS) teilnehmen. Anweisungen zur Bereitstellung des erforderlichen Backends und zum Erstellen der Anwendung finden Sie in den folgenden GitHub-Repositorys:

- iOS: https://github.com/aws-samples/amazon-ivs-real-time-for-ios-demo/
- Android: https://github.com/aws-samples/amazon-ivs-real-time-for-android-demo/
- Backend: https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

# Support

Das AWS-Supportcenter bietet eine Reihe von Plänen, die den Zugriff auf Tools und das Know-how zur Unterstützung Ihrer AWS-Lösungen bieten. Alle Supportpläne bieten Zugriff auf Kundenservice rund um die Uhr. Wenn Sie technischen Support und Zugriff auf zusätzliche Ressourcen benötigen, um Ihre AWS-Umgebung zu planen, bereitzustellen und zu verbessern, können Sie einen Support-Plan auswählen, der für Ihren AWS-Anwendungsfall optimal angepasst ist.

AWS Premium Support ist ein direkter und schneller Supportkanal, der Sie beim Entwickeln und Ausführen von Anwendungen in AWS unterstützt.

AWS re:Post ist eine Community-basierte Q&A-Website, die für Entwickler eingerichtet wurde, um über technische Fragen zu Amazon IVS zu diskutieren.

Kontakt Links für nicht-technische Anfragen zu Ihrer Abrechnung oder Ihrem Konto. Technische Fragen stellen Sie bitte in den Diskussionsforen oder über die Support-Links.

# Glossar

Weitere Informationen finden Sie im AWS-Glossar. In der folgenden Tabelle steht LL für IVS-Streaming mit niedriger Latenz. RT und IVS-Echtzeit-Streaming.

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| AAC | Erweiterte Audio-Kodierung. AAC ist ein Audio-Kodierungsstandard für verlustbehaftete digitale Audiokomprimierung. Als Nachfolger des MP3-Formats konzipiert, erreicht AAC bei gleicher Bitrate im Allgemeinen eine höhere Klangqualität als MP3. AAC wurde von ISO und IEC als Teil der Spezifikationen MPEG-2 und MPEG-4 standardi siert. | ✓ | ✓ | |
| Streaming mit adaptiver Bitrate | Beim Streaming mit adaptiver Bitrate (ABR) kann der IVS-Player auf eine niedrigere Bitrate umschalten, wenn die Verbindungsqualität beeinträchtigt ist, und auf eine höhere Bitrate zurückschalten, wenn sich die Verbindungsqualität verbessert. | ✓ | | |
| Adaptives Streaming | Weitere Informationen finden Sie unter Mehrschic htige Kodierung mit Simulcast. | | ✓ | |
| Administratorbenut zer | Ein AWS-Benutzer mit Administratorzugriff auf Ressourcen und Services, die in einem AWS-Konto verfügbar sind. Weitere Informationen finden Sie unter Terminologie im Benutzerhandbuch zur AWS-Einrichtung. | ✓ | ✓ | ✓ |
| ARN | Amazon-Ressourcenname, eine eindeutige Kennung für eine AWS-Ressource. Spezifisc he ARN-Formate hängen vom Ressourcentyp ab. Informationen zu den von IVS-Ressourcen | ✓ | ✓ | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| | verwendeten ARN-Formaten finden Sie in der Service-Autorisierungsreferenz. | | | |
| Seitenverhältnis | Beschreibt das Verhältnis der Rahmenbreite zur Rahmenhöhe. Beispielsweise ist 16:9 das Seitenverhältnis, das der Full-HD- oder 1080p-Auflösung entspricht. | ✓ | ✓ | |
| Audio-Modus | Eine voreingestellte oder benutzerdefinierte Audiokonfiguration, die für verschiedene Arten von Benutzern mobiler Geräte und die von ihnen verwendeten Geräte optimiert ist. Weitere Informationen finden Sie unter IVS Broadcast SDK: Mobile Audiomodi (Echtzeit-Streaming). | | ✓ | |
| AVC, H.264, MPEG-4 Teil 10 | Erweiterte Video-Kodierung, auch als H.264 oder MPEG-4 Teil 10 bezeichnet, ein Videokomprimierungsstandard für verlustbehaftete digitale Videokomprimierung. | ✓ | ✓ | |
| Ersetzen des Hintergrunds | Eine Art Kamerafilter, der es Livestream-Erstellern ermöglicht, ihren Hintergrund zu ändern. Weitere Informationen finden Sie unter Ersetzen des Hintergrunds in IVS Broadcast SDK: Kamera-Filter von Drittanbietern (Echtzeit-Streaming). | | ✓ | |
| Bitrate | Eine Streaming-Metrik für die Anzahl der pro Sekunde übertragenen oder empfangenen Bits. | ✓ | ✓ | |
| Broadcast, Sender | Andere Begriffe für Stream, Streamer. | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Pufferung | Ein Zustand, der auftritt, wenn das Wiedergab egerät den Inhalt nicht herunterladen kann, bevor der Inhalt abgespielt werden soll. Pufferung kann sich auf verschiedene Weise äußern: Inhalte können zufällig anhalten und starten (auch Stottern genannt), Inhalte können für längere Zeit anhalten (auch Einfrieren genannt) oder der IVS-Player kann die Wiedergabe anhalten. | ✓ | ✓ | |
| Wiedergabeliste im Bytebereich | Eine differenziertere Wiedergabeliste als die standardmäßige HLS-Wiedergabeliste. Die standardmäßige HLS-Wiedergabeliste besteht aus 10-sekündigen Mediendateien. Bei einer Wiedergabeliste mit Byte-Bereich entspricht die Segmentdauer dem Keyframe-Intervall, das für den Stream konfiguriert wurde.<br><br>Die Wiedergabeliste im Bytebereich ist nur für Übertragungen verfügbar, die automatisch in einem S3-Bucket aufgezeichnet wurden. Diese wird zusätzlich zur HLS-Wiedergabeliste erstellt. Weitere Informationen finden Sie unter Wiedergab elisten im Byte-Bereich in Automatische Aufzeichn ung in Amazon S3 (Streaming mit niedriger Latenz). | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| CBR | Konstante Bitrate, eine Ratensteuerungsmethode für Encoder, die während der gesamten Wiedergabe eines Videos auf eine einheitliche Bitrate warten, unabhängig davon, was während der Übertragung passiert. Tiefpunkte im Geschehen können aufgefüllt werden, um die gewünschte Bitrate zu erreichen, und Spitzen können quantisiert werden, indem die Qualität der Kodierung an die Ziel-Bitrate angepasst wird. Wir empfehlen nachdrücklich die Verwendung von CBR anstelle von VBR. | ✓ | ✓ | |
| CDN | Netzwerk für die Inhaltsbereitstellung oder Netzwerk für die Inhaltsübermittlung, eine geografisch verteilte Lösung, die die Bereitstellung von Inhalten wie Streaming-Videos optimiert, indem sie diese näher an den Standort der Benutzer bringt. | ✓ | | |
| Kanal | Eine IVS-Ressource, die die Konfiguration für das Streaming speichert, einschließlich eines Aufnahmeservers, eines Stream-Schlüssels, einer Wiedergabe-URL und Aufzeichnungsoptionen. Streamer verwenden den Streamschlüssel, der einem Kanal zugeordnet ist, um eine Übertragung zu starten. Alle während einer Übertragung generierten Metriken und Ereignisse werden einer Kanalressource zugeordnet. | ✓ | | |
| Kanaltyp | Legt die zulässige Auflösung und Bildfrequenz für den Kanal fest. Siehe Kanaltypen in der Referenz der API von IVS-Streaming mit niedriger Latenz. | ✓ | | |
| Chat-Protokollierung | Eine erweiterte Option, die durch die Zuordnung einer Protokollierungskonfiguration zu einem Chatroom ermöglicht wird. | | | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Chatroom | Eine IVS Ressource, die die Konfiguration für eine Chatsitzung speichert, einschließlich optionale r Features wie Handler zur Nachrichtenüberprü fung und Chat-Protokollierung. Weitere Informati onen finden Sie unter Schritt 2: Erstellen eines Chatrooms unter Erste Schritte mit Amazon IVS Chat. | | | ✓ |
| Clientseitige Zusammensetzung | Verwendet ein Host-Gerät zum Mischen von Audio- und Videostreams von Stufenteilnehmern und sendet sie dann als zusammengesetzten Stream an einen IVS-Kanal. Dies ermöglicht eine bessere Kontrolle über das Erscheinungsbild der Zusammensetzung, allerdings auf Kosten einer höheren Auslastung der Client-Ressourcen und eines höheren Risikos, dass sich ein Stufen- oder Host-Problem auf die Zuschauer auswirkt.<br><br>Weitere Informationen finden Sie unter serversei tige Zusammensetzung. | ✓ | ✓ | |
| CloudFront | Ein von Amazon bereitgestellter CDN-Service. | ✓ | | |
| CloudTrail | Ein AWS-Service zur Erfassung, Überwachung, Analyse und Beibehaltung von Ereignissen und Kontoaktivitäten von AWS und externen Quellen. Siehe Protokollieren von IVS-API-Aufrufen mit AWS CloudTrail. | ✓ | ✓ | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| CloudWatch | Ein AWS-Service zur Überwachung von Anwendungen, zur Reaktion auf Leistungsänderungen, zur Optimierung der Ressourcennutzung und zur Bereitstellung von Einblicken in den Betriebszustand. Sie können verwenden CloudWatch , um IVS-Metriken zu überwachen. Weitere Informationen finden Sie unter Überwachen von IVS-Echtzeit-Streaming und Überwachen von IVS-Streaming mit niedriger Latenz. | ✓ | ✓ | ✓ |
| Composition | Der Prozess des Zusammenführens von Audio- und Videostreams aus mehreren Quellen zu einem einzigen Stream. | ✓ | ✓ | |
| Pipeline der Zusammensetzung | Eine Abfolge von Verarbeitungsschritten, die zum Kombinieren mehrerer Streams und zum Kodieren des resultierenden Streams erforderlich sind. | ✓ | ✓ | |
| Komprimierung | Kodierung von Informationen mit weniger Bits als in der ursprünglichen Darstellung. Jede einzelne Komprimierung ist entweder verlustfrei oder verlustbehaftet. Die verlustfreie Komprimierung reduziert die Anzahl der Bits durch die Identifizierung und Beseitigung statistischer Redundanz. Bei der verlustfreien Komprimierung gehen keine Informationen verloren. Bei der verlustbehafteten Komprimierung werden Bits reduziert, indem unnötige oder weniger wichtige Informationen entfernt werden. | ✓ | ✓ | |
| Steuerebene | Speichert Informationen zu IVS-Ressourcen wie Kanälen, Stufen oder Chatrooms und stellt Schnittstellen zum Erstellen und Verwalten dieser Ressourcen bereit. Es ist regional (basierend auf AWS-Regionen). | ✓ | ✓ | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| CORS | Herkunftsübergreifende Ressourcenfreigabe, ein AWS-Feature, mit dem Client-Webanwendungen, die in einer Domain geladen sind, mit Ressourcen wie S3-Buckets in einer anderen Domain interagieren können. Der Zugriff kann basierend auf Headern, HTTP-Methoden und Ursprungsdomains konfiguriert werden. Weitere Informationen finden Sie unter Nutzung der herkunftsübergreifenden Ressourcennutzung (CORS) – Amazon Simple Storage Service im Benutzerhandbuch für Amazon Simple Storage Service. | ✓ | | |
| Benutzerdefinierte Bildquelle | Eine vom IVS Broadcast SDK bereitgestellte Schnittstelle, über die eine Anwendung ihre eigene Bildeingabe bereitstellen kann, anstatt auf die voreingestellten Kameras beschränkt zu sein. | ✓ | ✓ | |
| Datenebene | Die Infrastruktur, die Daten von der Aufnahme bis zum Ausgang überträgt. Der Betrieb basiert auf der in der Steuerebene verwalteten Konfiguration und ist nicht auf eine AWS-Region beschränkt. | ✓ | ✓ | ✓ |
| Encoder, Verschlüsselung | Der Vorgang der Konvertierung von Video- und Audioinhalten in ein für Streaming geeignetes digitales Format. Die Kodierung kann hardware- oder softwarebasiert sein. | ✓ | ✓ | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Ereignis | Eine automatische Benachrichtigung, die von IVS an den AmazonEventBridge Überwachungsservice veröffentlicht wird. Ein Ereignis stellt eine Zustands- oder Zustandsänderung einer Streaming-Ressourc e dar, beispielsweise einer Stufe oder einer Zusammensetzungs-Pipeline. Weitere Informati onen finden Sie unter Verwenden von Amazon EventBridge mit IVS-Streaming mit niedriger Latenz und Verwenden von Amazon EventBridge mit IVS-Echtzeit-Streaming. | ✓ | ✓ | ✓ |
| FFmpeg | Ein kostenloses Open-Source-Softwareprojekt, das aus einer Reihe von Bibliotheken und Programme n zur Verarbeitung von Video- und Audiodateien und -streams besteht. FFmpeg bietet eine plattform übergreifende Lösung zum Aufzeichnen, Konvertie ren und Streamen von Audio und Video. | ✓ | | |
| Fragmentierter Stream | Wird erstellt, wenn eine Übertragung innerhalb des in der Aufzeichnungskonfiguration des Kanals angegebenen Intervalls unterbrochen und dann erneut verbunden wird. Die resultier enden mehreren Streams werden als eine einzelne Übertragung betrachtet und zu einem einzigen aufgezeichneten Stream zusammengeführt. Weitere Informationen finden Sie unter Zusammenf ühren fragmentierter Streams in Automatische Aufzeichnung in Amazon S3 (Streaming mit niedriger Latenz). | ✓ | | |
| Bildrate | Eine Streaming-Metrik für die Anzahl der übertrage nen oder empfangenen Videobilder pro Sekunde. | ✓ | ✓ | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| HLS | HTTP Live Streaming (HLS), ein HTTP-basiertes Streaming-Kommunikationsprotokoll mit adaptiver Bitrate, das zur Bereitstellung von IVS-Streams an Zuschauer verwendet wird. | ✓ | | |
| HLS-Wiedergabeliste | Eine Liste von Mediensegmenten, aus denen ein Stream besteht. Standard-HLS-Wiedergabelisten bestehen aus 10-sekündigen Mediendateien. HLS unterstützt auch detailliertere Wiedergabelisten im Bytebereich. | ✓ | | |
| Host | Ein Echtzeit-Teilnehmer eines Ereignisses, der Video und/oder Audio an die Stufe sendet. | | ✓ | |
| IAM | Identity and Access Management, ein AWS-Service, der Benutzern die sichere Verwaltung von Identitäten und Zugriff auf AWS-Services und -Ressourcen, einschließlich IVS, ermöglicht. | ✓ | ✓ | ✓ |
| Ergest | IVS-Prozess zum Empfang von Videostreams von einem Host oder Sender zur Verarbeitung oder Bereitstellung an Zuschauer oder andere Teilnehmer. | ✓ | ✓ | |
| Server Ingest | Empfängt Videostreams und überträgt sie an ein Transkodierungssystem, wo Streams für die Bereitstellung an die Zuschauer in HLS transmuxiert oder transkodiert werden.<br><br>Aufnahme-Server sind spezielle IVS-Komponenten, die Streams für Kanäle zusammen mit einem Aufnahmeprotokoll (RTMP, RTMPS) empfangen. Weitere Informationen zum Erstellen eines Kanals finden Sie unter Erste Schritte mit IVS-Streaming mit niedriger Latenz. | | ✓ | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Video mit Zeilensprung | Überträgt und zeigt nur ungerade oder gerade Zeilen von aufeinanderfolgenden Frames an, um eine wahrgenommene Verdoppelung der Bildfrequenz zu erreichen, ohne zusätzliche Bandbreite zu verbrauchen. Aufgrund von Bedenken hinsichtlich der Videoqualität wird die Verwendung von Video mit Zeilensprung nicht empfohlen. | ✓ | ✓ | |
| JSON | JavaScript Object Notation, ein Open-Standard-Dateiformat, das von Menschen lesbaren Text verwendet, um Datenobjekte zu übertragen, die aus Attribut-Wert-Paaren und Array-Datentypen oder anderen serialisierbaren Werten bestehen. | ✓ | ✓ | ✓ |
| Keyframe, Delta-Frame, Keyframe-Intervall | Der Keyframe (auch als intra-kodiert oder i-Frame bezeichnet) ist ein Vollbild des Bildes in einem Video. Nachfolgende Frames, die Deltaframes (auch als prognostizierte oder p-Frames bezeichnet), enthalten nur die geänderten Informationen. Abhängig vom im Encoder definierten Keyframe-Intervall werden Keyframes innerhalb eines Streams mehrmals angezeigt. | ✓ | ✓ | |
| Lambda | Ein AWS-Service zum Ausführen von Code (als Lambda-Funktionen bezeichnet) ohne Bereitstellung einer Serverinfrastruktur. Lambda-Funktionen können als Reaktion auf Ereignisse und Aufrufanfragen oder basierend auf einem Zeitplan ausgeführt werden. IVS Chat verwendet beispielsweise Lambda-Funktionen, um die Nachrichtenüberprüfung für einen Chatroom zu ermöglichen. | ✓ | ✓ | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Latenz, glass-to-glass Latenz | Eine Verzögerung bei der Datenübertragung. IVS definiert Latenzbereiche wie folgt:<br>• Niedrige Latenz: unter 3 Sekunden<br>• Latenz in Echtzeit: unter 300 ms<br><br>Glatenzlass-to-glass bezieht sich auf die Verzögerung zwischen dem Zeitpunkt, an dem eine Kamera einen Livestream aufnimmt, und dem Zeitpunkt, an dem der Stream auf dem Bildschirm eines Viewers angezeigt wird. | ✓ | ✓ | |
| Mehrschichtige Kodierung mit Simulcast | Ermöglicht die gleichzeitige Kodierung und Veröffentlichung mehrerer Videostreams mit unterschiedlichen Qualitätsstufen. Weitere Informationen finden Sie unter Adaptives Streaming : Mehrschichtige Kodierung mit Simulcast in Streaming-Optimierungen in Echtzeit. | | ✓ | |
| Handler für Nachrichtenüberprüfung | Ermöglicht es IVS-Chat-Kunden, Benutzer-Chat-Nachrichten automatisch zu überprüfen/zu filtern, bevor sie an den Chatroom übermittelt werden. Dies wird durch die Verknüpfung einer Lambda-Funktion mit einem Chatroom ermöglicht. Weitere Informationen finden Sie unter Erstellen einer Lambda-Funktion im Handler für Nachrichtenüberprüfung. | | | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Mischpult | Ein Feature der IVS Mobile Broadcast SDKs, die mehrere Audio- und Videoquellen aufnimmt und eine einzige Ausgabe generiert. Diese Funktion unterstützt die Verwaltung von Video- und Audioelementen auf dem Bildschirm, die Quellen wie Kameras, Mikrofone, Bildschirmaufnahmen sowie von der Anwendung generiertes Audio und Video darstellen. Die Ausgabe kann anschließend an IVS gestreamt werden. Weitere Informationen finden Sie unter Konfigurieren einer Broadcast -Sitzung zum Mischen im IVS Broadcast SDK: Mixer-Handbuch (Streaming mit niedriger Latenz). | ✓ | | |
| Streaming auf mehreren Hosts | Kombiniert Streams von mehreren Hosts zu einem einzigen Stream. Dies kann entweder durch clientseitige oder serverseitige Zusammensetzung erreicht werden.<br><br>Das Streaming mit mehreren Hosts ermöglicht Szenarien wie die Einladung von Zuschauern auf eine Stufe für Fragen und Antworten, Wettbewer be zwischen Hosts, Videochats und Gespräche zwischen den Hosts vor einem großen Publikum. | | ✓ | |
| Multivariante Wiedergabeliste | Ein Index aller Varianten-Streams, die für eine Übertragung verfügbar sind. | ✓ | | |
| OAC | Origin Access Control, ein Mechanismus zur Einschränkung des Zugriffs auf einen S3-Bucket , sodass Inhalte wie ein aufgezeichneter Stream nur über CloudFront CDN bereitgestellt werden können. | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| OBS | Open Broadcaster Software, kostenlose und Open-Source-Software für Videoaufzeichnung und Live-Streaming. OBS bietet eine Alternative (zum IVS Broadcast SDK) für Desktop-Publishing. Erfahrenere Streamer, die mit OBS vertraut sind, bevorzugen es möglicherweise aufgrund seiner erweiterten Produktionsfeatures wie Szenenübergänge, Audiomischung und Overlay-Grafiken. | ✓ | ✓ | |
| Teilnehmer | Ein Echtzeitbenutzer, der als Host oder Zuschauer mit einer Stufe verbunden ist. | | ✓ | |
| Teilnehmer-Token | Authentifiziert einen Teilnehmer eines Ereignisses in Echtzeit, wenn er einer Stufe beitritt. Ein Teilnehmer-Token steuert auch, ob ein Teilnehmer Videos an die Stufe senden kann. | | ✓ | |
| Wiedergabe-Token, Wiedergabe-Schlüsselpaar | Ein Autorisierungsmechanismus, mit dem Kunden die Videowiedergabe auf privaten Kanälen beschränken können. Wiedergabe-Token werden aus einem Wiedergabe-Schlüsselpaar generiert.<br><br>Ein Wiedergabe-Schlüsselpaar ist das öffentlich-private Schlüsselpaar, das zum Signieren und Validieren des Viewer-Autorisierungs-Token für die Wiedergabe verwendet wird. Weitere Informationen finden Sie unter Erstellen oder Importieren eines Wiedergabeschlüssels unter Einrichtung privater Kanäle. Weitere Informationen zu den Endpunkten für Wiedergabeschlüsselpaare finden Sie in der IVS-API-Referenz für niedrige Latenz. | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Wiedergabe-URL | Gibt die Adresse an, die ein Zuschauer verwendet , um die Wiedergabe für einen bestimmten Kanal zu starten. Diese Adresse kann global verwendet werden. Amazon IVS wählt automatisch den besten Standort im globalen Netzwerk für die Inhaltsbereitstellung von IVS aus, um das Video an jeden Zuschauer zu übermitteln. Weitere Informationen zum Erstellen eines Kanals finden Sie unter Erste Schritte mit IVS-Streaming mit niedriger Latenz. | ✓ | | |
| Privater Kanal | Ermöglicht Kunden die Einschränkung des Zugriffs auf ihre Streams mithilfe eines Autorisierungsmechanismus, der auf Wiedergabe-Tokens basiert. Weitere Informationen finden Sie unter Workflow für private Kanäle unter Einrichten privater Kanäle. | ✓ | | |
| Progressives Video | Überträgt und zeigt alle Zeilen jedes Frames der Reihe nach an. Es empfiehlt sich die Verwendung von progressivem Video in allen Phasen einer Übertragung. | ✓ | ✓ | |
| Kontingente | Die maximale Anzahl von IVS-Serviceressourcen oder -Vorgängen für Ihr AWS-Konto. Das heißt, diese Grenzwerte gelten pro AWS-Konto, sofern nicht anders angegeben. Alle Kontingente werden pro Region erzwungen. Weitere Informationen finden Sie unter Endpunkten und Kontingenten von Amazon Interactive Video Service im Allgemeinen AWS-Referenzhandbuch. | ✓ | ✓ | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Regionen | Bieten Sie Zugriff auf AWS-Services, die sich physisch in einem bestimmten geografischen Gebiet befinden. Regionen bieten Fehlertoleranz, Stabilität und Ausfallsicherheit und können auch die Latenz verkürzen. Mit Regionen können Sie redundante Ressourcen erstellen, die verfügbar bleiben und von einem regionalen Ausfall nicht betroffen werden.<br><br>Die meisten AWS-Serviceanfragen beziehen sich auf eine bestimmte geografische Region. Die Ressourcen, die Sie in einer Region erstellen, sind in keiner anderen Region vorhanden, es sei denn, Sie verwenden ausdrücklich ein Replikati onsfeature, die von einem AWS-Service angeboten wird. Beispielsweise unterstützt Amazon S3 die regionsübergreifende Replikation. Einige Services, wie etwa IAM, verfügen über keine regionsüb ergreifende Ressourcen. | ✓ | ✓ | ✓ |
| Auflösung | Beschreibt die Anzahl der Pixel in einem einzelnen Videobild. Full HD oder 1080p definiert beispiels weise ein Frame mit 1920x1080 Pixeln. | ✓ | ✓ | |
| Stammbenutzer | Der Besitzer eines AWS-Kontos. Der Root-Benu tzer hat vollständigen Zugriff auf alle AWS-Services und Ressourcen im AWS-Konto. | ✓ | ✓ | ✓ |
| RTMP, RTMPS | Real-Time Messaging Protocol, ein Branchens tandard zum Übertragen von Audio, Video und Daten über ein Netzwerk. RTMPS ist die sichere Version von RTMP, die über eine Transport Layer Security (TLS/SSL)-Verbindung ausgeführt wird. | ✓ | ✓ | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| S3-Bucket | Eine Sammlung von Objekten, die in Amazon S3 gespeichert sind. Viele Richtlinien, einschließlich Zugriff und Replikation, werden auf Bucket-Ebene definiert und gelten für alle Objekte im Bucket. Beispielsweise wird eine IVS-Übertragung in Form mehrerer Objekte in einem S3-Bucket gespeichert. | ✓ | | |
| SDK | Software Development Kit, eine Sammlung von Bibliotheken für Entwickler, die Anwendungen mit IVS erstellen. | ✓ | ✓ | ✓ |
| Selfie-Segmentation | Ermöglicht das Ersetzen des Hintergrunds in einem Live-Stream mithilfe einer Client-spezifische n Lösung. Diese akzeptiert ein Kamerabild als Eingabe und gibt eine Maske zurück, die für jedes Pixel des Bildes eine Wertung bereitste llt und angibt, ob es sich im Vordergrund oder im Hintergrund befindet. Weitere Informationen finden Sie unter Ersetzen des Hintergrunds in IVS Broadcast SDK: Kamera-Filter von Drittanbietern (Echtzeit-Streaming). | | ✓ | |
| Semantische Versionsverwaltung | Ein Versionsformat in Form von Major.Minor.Patch. Fehlerkorrekturen, die sich nicht auf die API auswirken, erhöhen die Patch-Version, rückwärts kompatible API-Ergänzungen/Änderungen erhöhen die Nebenversion und rückwärtsinkompatible API-Änderungen erhöhen die Hauptversion. | ✓ | ✓ | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Serverseitige Zusammensetzung | Verwendet einen IVS-Server zum Mischen von Audio und Video von Teilnehmern einer Stufe und sendet dieses gemischte Video dann an einen IVS-Kanal, um ein größeres Publikum zu erreichen oder um es in einem S3-Bucket zu speichern. Die serverseitige Zusammensetzung reduziert die Client-Auslastung, verbessert die Stabilität der Übertragung und ermöglicht eine effizientere Nutzung der Bandbreite.<br><br>Weitere Informationen finden Sie unter Clientseitige Zusammensetzung. | | ✓ | |
| Service Quotas | Ein AWS-Service, mit dem Sie Ihre Kontingente für viele AWS-Services von einem Standort aus verwalten können. Sie können über die Service–Quotas-Konsole Kontingentwerte abfragen und außerdem Kontingenterhöhungen anfordern. | ✓ | ✓ | ✓ |
| Servicegebundene Rolle | Ein eindeutiger IAM-Rollentyp, der direkt mit einem AWS-Service verknüpft ist. Serviceverknüpfte Rollen werden automatisch von IVS erstellt und enthalten alle Berechtigungen, die der Service benötigt, um andere AWS-Services in Ihrem Namen aufzurufen, z. B. für den Zugriff auf einen S3-Bucket. Weitere Informationen finden Sie unter Nutzung serviceverknüpfter Rollen für IVS in IVS-Sicherheit. | ✓ | | |
| Stufe | Eine IVS Ressource, die eine virtuelle Umgebung darstellt, in dem die Echtzeit-Teilnehmer eines Ereignisses Videos in Echtzeit austauschen können. Weitere Informationen finden Sie unter Erstellen einer Stufe in Erste Schritte mit IVS-Echtzeit-Streaming. | | ✓ | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Stufensitzung | Beginnt, wenn der erste Teilnehmer eine Stufe betritt und endet einige Minuten, nachdem der letzte Teilnehmer die Veröffentlichung in der Stufe beendet hat. Eine langlebige Stufe kann im Laufe ihrer Lebensdauer möglicherweise mehrere Sitzungen haben. | | ✓ | |
| Stream | Daten, die Video- oder Audioinhalte darstellen und fortlaufend von einer Quelle an ein Ziel gesendet werden. | ✓ | ✓ | |
| Stream-Schlüssel | Eine von IVS beim Erstellen eines Kanals zugewiesene Kennung. Es wird verwendet, um das Streaming zum Kanal zu autorisieren. Behandeln Sie den Stream-Schlüssel wie ein Geheimnis, da jeder mit ihm berechtigt ist auf den Kanal zu streamen.. Weitere Informationen finden Sie unter Erste Schritte mit IVS-Streaming mit niedriger Latenz. | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Stream-Starvation | Eine Verzögerung oder ein Stopp bei der Stream-Übermittlung an IVS. Dies tritt auf, wenn IVS nicht die erwartete Anzahl an Bits empfängt, die das Kodierungsgerät angekündigt hat und die es über einen bestimmten Zeitraum senden würde. Das Auftreten einer Stream-Starvation führt zu einem Stream-Starvation-Ereignis.<br><br>Aus der Sicht eines Zuschauers kann eine Stream-Starvation als verzögertes, pufferndes oder einfrierendes Video erscheinen. Die Stream-St arvation kann kurz (weniger als 5 Sekunden) oder lang (mehrere Minuten) sein, abhängig von der spezifischen Situation, die zur Stream-Starvation geführt hat. Weitere Informationen finden Sie unter Was ist Stream-Starvation in den Häufig gestellten Fragen zur Fehlerbehebung. | ✓ | ✓ | |
| Streamer | Eine Person oder ein Gerät, das einen Video- oder Audio-Stream an IVS sendet. | ✓ | ✓ | |
| Subscriber | Ein Teilnehmer eines Ereignisses in Echtzeit, der Video- und/oder Audioaufnahmen der Hosts empfängt. Weitere Informationen finden Sie unter Was ist IVS-Echtzeit-Streaming?. | | ✓ | |
| Markierung | Ein Tag ist eine Markierung, die Sie einer AWS-Ressource zuordnen. Mithilfe von Tags können Sie Ihre AWS-Ressourcen leichter identifizieren und anordnen. Auf der Startseite der IVS-Dokum entation finden Sie den Abschnitt „Tagging“ in einer beliebigen IVS-API-Dokumentation (für Echtzeit-Streaming, Streaming mit niedriger Latenz oder Chat). | ✓ | ✓ | ✓ |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Kamerafilter von Drittanbietern | Softwarekomponenten, die in das IVS Broadcast SDK integriert werden können, damit eine Anwendung Bilder verarbeiten kann, bevor sie dem Broadcast SDK als benutzerdefinierte Image-Quelle bereitgestellt werden. Ein Kamerafilter eines Drittanbieters kann Bilder von der Kamera verarbeiten, einen Filtereffekt anwenden usw. | ✓ | ✓ | |
| Miniaturansicht | Ein verkleinertes Bild, das aus einem Stream aufgenommen wurde. Standardmäßig werden Miniaturansichten alle 60 Sekunden generiert, es kann jedoch ein kürzeres Intervall konfiguriert werden. Die Auflösung der Miniaturansicht hängt vom Kanaltyp ab. Weitere Informationen finden Sie unter Aufzeichnen von Inhalten in Automatische Aufnahme in Amazon S3 (Streaming mit niedriger Latenz). | ✓ | | |
| Zeitgesteuerte Metadaten | An bestimmte Zeitstempel innerhalb eines Streams gebundene Metadaten. Dies kann programmgesteuert mithilfe der IVS-API hinzugefügt werden und wird bestimmten Frames zugeordnet. Dadurch wird sichergestellt, dass alle Zuschauer die Metadaten an der gleichen Stelle relativ zum Stream erhalten.<br><br>Zeitgesteuerte Metadaten können verwendet werden, um Aktionen auf dem Client auszulösen, z. B. die Aktualisierung von Teamstatistiken während einer Sportveranstaltung. Weitere Informationen finden Sie unter Einbettung von Metadaten in einen Video-Stream. | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| Transkodierung | Wandelt Video und Audio von einem Format in ein anderes um. Ein eingehender Stream kann in ein anderes Format mit mehreren Bitraten und Auflösungen transkodiert werden, um eine Reihe von Wiedergabegeräten und Netzwerkbedingungen zu unterstützen. | ✓ | ✓ | |
| Transmuxing | Ein einfaches Umpacken eines aufgenommenen Streams in IVS ohne erneute Kodierung des Videostreams. „Transmux“ ist die Abkürzung für Transcode-Multiplexing, ein Prozess, der das Format einer Audio- und/oder Videodatei ändert und dabei einige oder alle der ursprünglichen Streams beibehält. Transmuxing konvertiert in ein anderes Containerformat, ohne den Dateiinhalt zu ändern. Unterscheidet sich von Transkodierung. | ✓ | ✓ | |
| Varianten-Streams | Eine Reihe von Kodierungen derselben Übertragung in verschiedenen Qualitätsstufen. Jeder Varianten-Stream wird als separate HLS-Wiedergabeliste kodiert. Ein Index der verfügbaren Varianten-Streams wird als multivariante Wiedergabeliste bezeichnet.<br><br>Nachdem der IVS-Player eine multivariante Playlist von IVS empfangen hat, kann er während der Wiedergabe zwischen den Varianten-Streams auswählen und bei sich ändernden Netzwerkbedingungen nahtlos hin und her wechseln. | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| VBR | Variable Bitrate, eine Methode der Ratensteuerung für Encoder, die eine dynamische Bitrate verwendet, die sich während der Wiedergabe je nach erforderlicher Detailebene ändert. Aus Gründen der Videoqualität raten wir nachdrücklich davon ab, VBR zu verwenden. Verwenden Sie stattdessen CBR. | ✓ | ✓ | |
| Anzeigen | Eine einzigartige Anzeigesitzung, die aktiv Videos herunterlädt oder abspielt. Aufrufe sind die Grundlage für das Kontingent gleichzeitiger Aufrufe.<br><br>Eine Ansicht beginnt, wenn eine Anzeigesitzung die Videowiedergabe beginnt. Eine Ansicht endet, wenn eine Anzeigesitzung die Videowiedergabe stoppt. Die Wiedergabe ist der einzige Indikator für die Zuschauerschaft; Interaktionsheuristiken wie Audiopegel, Browser-Tab-Fokus und Videoqualität werden nicht berücksichtigt. Beim Zählen der Aufrufe berücksichtigt IVS nicht die Legitimität einzelner Zuschauer und versucht auch nicht, lokalisierte Zuschauerzahlen zu deduplizieren, z. B. mehrere Videoplayer auf einem einzigen Computer. Weitere Informationen finden Sie unter Andere Kontingente in Service Quotas (Streaming mit niedriger Latenz). | ✓ | | |
| Zuschauer | Eine Person, die einen Stream von IVS empfängt. | ✓ | | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| WebRTC | Web Real-Time Communication, ein Open-Source-Projekt, das Webbrowsern und mobilen Anwendungen Echtzeitkommunikation bietet. Es ermöglicht die Audio- und Videokommunikation, um innerhalb von Webseiten zu arbeiten, indem es eine direkte peer-to-peer Kommunikation ermöglicht, sodass keine Plug-Ins installiert oder native Apps heruntergeladen werden müssen.<br><br>Die Technologien hinter WebRTC werden als offener Webstandard implementiert und sind als reguläre JavaScript APIs in allen großen Browsern oder als Bibliotheken für native Clients wie Android und iOS verfügbar. | ✓ | ✓ | |

| Begriff | Beschreibung | LL | RT | Chat |
|---|---|---|---|---|
| WHIP | WebRTC-HTTP Ingestion Protocol, ein HTTP-basiertes Protokoll, das die WebRTC-basierte Aufnahme von Inhalten in Streaming-Services und/oder CDNs ermöglicht. WHIP ist ein IETF-Entwurf, der zur Standardisierung der WebRTC-Erfassung entwickelt wurde.<br><br>WHIP ermöglicht die Kompatibilität mit Software wie OBS und bietet eine Alternative (zum IVS-Broadcast-SDK ) für die Desktop-Veröffentlichung. Versierte Streamer, die mit OBS vertraut sind, bevorzugen es möglicherweise aufgrund ihrer erweiterten Produktionsfunktionen wie Szenenübergänge, Audiomix und Overlay-Grafiken<br><br>WHIP ist auch in Situationen von Vorteil, in denen die Verwendung des IVS-Broadcast-SDK nicht möglich oder bevorzugt ist. Beispielsweise ist das IVS-Broadcast-SDK in Setups mit Hardware-Encodern möglicherweise keine Option. Wenn der Encoder jedoch WHIP unterstützt, können Sie trotzdem direkt vom Encoder in IVS veröffentlichen.<br><br>Siehe OBS- und WHIP-Unterstützung. | | ✓ | |
| WSS | WebSocket Secure, ein Protokoll zum Herstellen WebSockets einer verschlüsselten TLS-Verbindung. Dies wird zum Herstellen einer Verbindung mit IVS-Chat-Endpunkten verwendet. Weitere Informationen finden Sie unter Schritt 4: Senden und Empfangen Ihrer ersten Nachricht in Erste Schritte mit IVS-Chat. | | | ✓ |

# Dokumentenverlauf (Echtzeit-Streaming)

## Änderungen im Benutzerhandbuch für Echtzeit-Streaming

| Änderung | Beschreibung | Datum |
| --- | --- | --- |
| OBS- und WHIP-Unterstützung | Neue Seite hinzugefügt. In diesem Dokument wird erläutert, wie Sie WHIP-kompatible Encoder wie OBS verwenden, um in IVS-Echtzeit-Streaming zu veröffentlichen. WHIP (WebRTC-HTTP Ingestion Protocol) ist ein IETF-Entwurf, der zur Standardisierung der WebRTC-Erfassung entwickelt wurde. | 6. Februar 2024 |
| Broadcast-SDK: Android 1.14.1, iOS 1.14.1, Web 1.8.0 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version in den real-time-streaming Broadcast-SDK-Anleitungen: Android, iOS und Web. Auf der Startseite für die Amazon-IVS-Dokumentation wurden die Broadcast-SDK-Referenzlinks mit Verweisen auf die neue Version aktualisiert. Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version.<br><br>Für das Android-Handbuch haben wir ein neues bekanntes Problem hinzugefü | 1. Februar 2024 |

| | | |
|---|---|---|
| | gt (Videogröße unter 176x176).<br><br>Für das Webhandbuch haben wir ein neues bekanntes Problem hinzugefügt. Die Problemumgehung beschränkt die Videoauflösung beim Aufrufen von `getUserMedia` oder auf 720p`getDisplayMedia` .<br><br>In Echtzeit-Streaming-Optimierungen haben wir die Konfiguration der mehrschichtigen Kodierung mit Simulcast aktualisiert. Dies ist jetzt standardmäßig deaktiviert. | |
| Broadcast-SDK: Android 1.13.4, iOS 1.13.4, Web 1.7.0 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version in den real-time-streaming Broadcast-SDK-Anleitungen: Android, iOS und Web . Auf der Startseite für die Amazon-IVS-Dokumentation wurden die Broadcast-SDK-Referenzlinks mit Verweisen auf die neue Version aktualisiert. Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version. | 3. Januar 2024 |
| IVS-Glossar | Das Glossar wurde um IVS-Begriffe in Echtzeit, niedriger Latenz und Chat erweitert. | 20. Dezember 2023 |

| | | |
|---|---|---|
| Bühnenzustand: Neue CloudWatch Metriken | Die PacketLoss Metrik (Stufe) wurde in DownloadPacketLoss (Stufe) umbenannt und zusätzliche CloudWatch Metriken für IVS-Echtzeit-Streaming veröffentlicht:<br>• DownloadPacketLoss (Stufe, Teilnehmer)<br>• DroppedFrames (Stufe, Teilnehmer)<br>• SubscribeBitrate (Stufe, Teilnehmer,MediaType)<br>Siehe Überwachen von IVS-Echtzeit-Streaming. | 07. Dezember 2023 |
| IAM-verwaltete Richtlinien | Zwei verwaltete Richtlinien wurden hinzugefügt, IVS ReadOnlyAccess und IVS FullAccess. Siehe:<br>• Der neue Abschnitt zu Verwaltete Richtlinien für Amazon IVS auf der Seite Sicherheit.<br>• Änderungen an Schritt 3: Einrichten von IAM-Berechtigungen in Erste Schritte mit IVS-Streaming mit niedriger Latenz. | 05. Dezember 2023 |

| | | |
|---|---|---|
| Broadcast-SDK: Android 1.13.2, iOS 1.13.2 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version in den real-time-streaming Broadcast-SDK-Anleitungen: Android und iOS .<br><br>Auf der Startseite für die Amazon-IVS-Dokumentation wurden die Broadcast-SDK-Referenzlinks mit Verweisen auf die neue Version aktualisiert.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version. | 4. Dezember 2023 |
| Broadcast-SDK: Android 1.13.1 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version im real-time-streaming Broadcast-SDK-Handbuch: Android .<br><br>Auf der Startseite für die Amazon-IVS-Dokumentation wurden die Broadcast-SDK-Referenzlinks mit Verweisen auf die neue Version aktualisiert.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version. | 21. November 2023 |

| | | |
|---|---|---|
| Service Quotas | Die „Auflösung der Veröffentlichung durch Teilnehmer“ wurde von 1080 p auf 720 p geändert. | 18. November 2023 |
| Broadcast-SDK: Android 1.13.0, iOS 1.13.0 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version in den real-time-streaming Broadcast-SDK-Anleitungen: Android und iOS .<br><br>Auf der Startseite für die Amazon-IVS-Dokumentation wurden die Broadcast-SDK-Referenzlinks mit Verweisen auf die neue Version aktualisiert.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version.<br><br>Es wurden außerdem verschiedene Aktualisierungen der Streaming-Optimierungen vorgenommen. Unter anderem erfordert das Feature „Adaptives Streaming: Ebenen-Codierung mit Simulcast“ jetzt eine ausdrückliche Zustimmung und wird nur in neueren Versionen des SDK unterstützt. | 17. November 2023 |

| | | |
|---|---|---|
| Zusammengesetzte Aufzeichnung | Durchführung der folgenden Änderungen:<br>• Für dieses neue Feature wurde eine Seite für zusammengesetzte Aufzeichnungen hinzugefügt.<br>• Aktualisierte Erste Schritte mit IVS-Echtzeit-Streaming mit S3-Endpunkten in der Richtlinie unter „IAM-Berechtigungen einrichten“.<br>• Aktualisierte Service Quotas mit Anrufratenkontingenten für die neuen Endpunkte. | 16. November 2023 |

| | | |
|---|---|---|
| Serverseitige Zusammensetzung (SSC) | Mit der serverseitigen IVS-Zusammensetzung können Clients die Zusammensetzung und Übertragung einer IVS-Stufe an einen von IVS verwalteten Service verlagern. SSC- und RTMP-Übertragungen an einen Kanal werden über IVS-Steuerebenen-Endpunkte in der Heimatregion der Stufe aufgerufen. Siehe:<br>• Erste Schritte – Der Richtlinie wurden unter „IAM-Berechtigungen einrichten“ SSC-Endpunkte hinzugefügt.<br>• Verwenden von Amazon EventBridge mit IVS – Wir haben neue Metriken hinzugefügt.<br>• Serverseitige Zusammensetzung – Dieses neue Dokument enthält eine Übersicht und Anweisungen zur Einrichtung.<br>• Service Quotas – Es wurden neue Anrufratenlimits und andere Kontingente hinzugefügt.<br><br>Lesen Sie auch:<br>• Unten aufgelistete Änderungen in der API- | 16. November 2023 |

| | | |
|---|---|---|
| | Referenz-Änderungen zu IVS-Echtzeit-Streaming.<br>• Aufgelistete Änderungen im Dokumentverlauf (Streaming mit niedriger Latenz). | |
| IVS-Broadcast-SDK | In der Broadcast-SDK-Übersicht wurde „Plattformanforderungen“ > „Native Plattformen“ aktualisiert, um klarzustellen, welche SDK-Versionen unterstützt werden. Außerdem wurden „Mobile Browser (iOS und Android)“ hinzugefügt.<br><br>Im Broadcast-Web-Leitfaden wurden „Einschränkungen für das mobile Web“ hinzugefügt. | 9. November 2023 |
| IVS-Broadcast-SDK | Eine neue Seite zu Kamerafiltern von Drittanbietern wurde hinzugefügt. | 9. November 2023 |
| Erste Schritte mit IVS-Echtzeit-Streaming | Wir haben die Verfahren im Abschnitt IAM-Berechtigungen einrichten aktualisiert. | 20. Oktober 2023 |
| Überwachen von Echtzeit-Streaming | In CloudWatch Metriken: IVS-Echtzeit-Streaming haben wir Beispielwerte für Dimensionen hinzugefügt. | 17. Oktober 2023 |
| Broadcast-SDK: Web-Leitfaden | Wir haben mehrere Änderungen an Überwachen des Medien-Stummschaltungsstatus von Remote-Teilnehmern vorgenommen. | 17. Oktober 2023 |

| | | |
|---|---|---|
| Broadcast-SDK: Web 1.6.0 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version im real-time-streaming Broadcast-SDK-Handbuch: Web .<br><br>Die Startseite der Amazon-IVS-Dokumentation verweist auf die aktuelle Version von Broadcast-SDK-Referenzen.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version.<br><br>Im Webhandbuch haben wir unter „Abrufen eines MediaStream von einem Gerät“ auch die beiden `max` Zeilen gelöscht. Die bewährte Methode besteht darin, nur anzugeben`ideal`.<br><br>Unter Echtzeit-Streaming-Optimierungen haben wir einen neuen Abschnitt hinzugefügt: Optimieren der Audio-Bitrate und der Stereo-Unterstützung. | 16. Oktober 2023 |
| Bühnenzustand: Neue CloudWatch Metriken | Veröffentlichte CloudWatch Metriken für IVS-Echtzeit-Streaming. Siehe Überwachen von IVS-Echtzeit-Streaming. | 12. Oktober 2023 |

| | | |
|---|---|---|
| Broadcast-SDK: Android 1.12.1 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version im real-time-streaming Broadcast-SDK-Handbuch: Android . Außerdem wurde ein neuer Abschnitt hinzugefügt: Verwenden von Bluetooth-Mikrofonen.<br><br>Die Startseite der Amazon-IVS-Dokumentation verweist auf die aktuelle Version von Broadcast-SDK-Referenzen.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version. | 12. Oktober 2023 |
| Broadcast-SDK: Web 1.5.2 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version im real-time-streaming Broadcast-SDK-Handbuch: Web .<br><br>Die Startseite der Amazon-IVS-Dokumentation verweist auf die aktuelle Version von Broadcast-SDK-Referenzen.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version. | 14. September 2023 |
| Erste Schritte mit IVS-Echtzeit-Streaming | Unter Android > Broadcast-SDK installieren wurde Datenbindung hinzugefügt. | 12. September 2023 |

| | | |
|---|---|---|
| Broadcast-SDK-Fehlerbehandlung | Die Abschnitte „Fehlerbehandlung“ wurden den Broadcast-SDK-Handbüchern hinzugefügt: Web, Android und iOS. | 12. September 2023 |
| Erste Schritte mit IVS-Echtzeit-Streaming | In Verteilen von Teilnehmertoken, wurde ein Wichtig-Hinweis hinzugefügt, der darüber informiert, dass Funktionen nicht auf dem aktuellen Tokenformat basieren. | 1. September 2023 |
| Erste Schritte mit IVS-Echtzeit-Streaming | In IAM-Berechtigungen einrichten wurde der Satz von Berechtigungen aktualisiert. | 31. August 2023 |
| Broadcast-SDK: Web 1.5.1, Android 1.12.0, und iOS 1.12.0 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version in den real-time-streaming Broadcast-SDK-Anleitungen: Web , Android und iOS .<br><br>Auf der Startseite für die Amazon-IVS-Dokumentation wurden die Broadcast-SDK-Referenzlinks mit Verweisen auf die neue Version aktualisiert.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version. | 23. August 2023 |

| | | |
|---|---|---|
| Start von Echtzeit-Streaming | Diese Version enthält wichtige Änderungen an der Dokumentation. Wir haben die vorherige Dokumentation in IVS-Streaming mit niedriger Latenz umbenannt und eine neue Dokumentation zu IVS-Echtzeit-Streaming veröffentlicht. Die Landingpage zur IVS-Dokumentation hat jetzt separate Abschnitte für Echtzeit-Streaming und Streaming mit niedriger Latenz. Jeder Abschnitt hat sein eigenes Benutzerhandbuch und eine eigene API-Referenz.<br><br>Weitere Änderungen an der Dokumentation finden Sie unter Dokumentenverlauf (Streaming mit niedriger Latenz). | 7. August 2023 |

| | | |
|---|---|---|
| Broadcast-SDK: Web 1.5.0, Android 1.11.0, und iOS 1.11.0 | Aktualisierte Versionsnummern und Artefakt-Links für die neue Version in den Broadcast-SDK-Anleitungen: Web, Android und iOS.<br><br>Auf der Startseite für die Amazon-IVS-Dokumentation wurden die Broadcast-SDK-Referenzlinks mit Verweisen auf die neue Version aktualisiert.<br><br>Sehen Sie auch die Versionshinweise von Amazon IVS für diese Version. | 7. August 2023 |

# Referenzänderungen an der API von IVS-Echtzeit-Streaming

| API-Änderungen | Beschreibung | Datum |
|---|---|---|
| Zusammengesetzte Aufzeichnung | Wir haben 4 StorageConfiguration Endpunkte und 7 Objekte (DestinationDetail, RecordingConfiguration, S3DestinationConfiguration, S3Detail, S3StorageConfiguration, StorageConfiguration) hinzugefügt StorageConfigurationSummary.<br><br>Wir haben 3 Objekte geändert (Komposition, Ziel, DestinationConfiguration). Dies wirkt sich auf die GetComposition Antwort sowie die StartComposition Anforderung und Antwort aus. | 16. November 2023 |
| Serverseitige Zusammensetzung | Wir haben 8 Zusammensetzungen und EncoderConfiguration Endpunkte und 11 Objekte (ChannelID estinationConfiguration, Zusammensetzung, | 16. November 2023 |

| API-Änderungen | Beschreibung | Datum |
| --- | --- | --- |
| | CompositionSummaryZiel, DestinationConfigu ration DestinationSummary, , EncoderConfiguration EncoderConfigurationSummary GridConfiguration LayoutConfiguration, und Video) hinzugefügt. | |
| Stufenzustand: Neue Teilnehmerdaten | Dem Objekt Teilnehmer wurden sechs Felder hinzugefügt: `browserName` , `browserVersion` , `ispName`, `osName`, `osVersion` , und `sdkVersio n` . Dies wirkt sich auf die GetParticipant Antwort aus. | 12. Oktober 2023 |
| Teilnehmertoken | Es wurde ein Wichtig-Hinweis hinzugefügt, der darüber informiert, dass Funktionen nicht auf dem aktuellen Tokenformat basieren. | 1. September 2023 |
| Start von IVS-Echtzeit-Streaming | Diese Version enthält wichtige Änderungen an der Dokumentation. Wir haben die vorherige Dokumenta tion in IVS-Streaming mit niedriger Latenz umbenannt und eine neue Dokumentation zu IVS-Echtzeit-Strea ming veröffentlicht. Die Landingpage zur IVS-Dokum entation hat jetzt separate Abschnitte für Echtzeit-Streaming und Streaming mit niedriger Latenz. Jeder Abschnitt hat sein eigenes Benutzerhandbuch und eine eigene API-Referenz.<br><br>Die Referenz zur API von IVS-Echtzeit-Streaming- ist Teil der IVS-Echtzeit-Streaming-Dokumentation. Zuvor trug sie den Titel Stage-API-Referenz für IVS. Ihre Vorgeschichte ist beschrieben in Dokumente nverlauf (Streaming mit niedriger Latenz). | 7. August 2023 |

# Versionshinweise (Echtzeit-Streaming)

## 6. Februar 2024

### OBS- und WHIP-Unterstützung

IVS kann mit WHIP-kompatiblen Encodern wie OBS verwendet werden, um in IVS-Echtzeit-Streaming zu veröffentlichen. WHIP (WebRTC -HTTP Ingestion Protocol) ist ein IETF-Entwurf, der zur Standardisierung der WebRTC-Erfassung entwickelt wurde. Weitere Informationen finden Sie auf der neuen Seite zu OBS und WHIP-Unterstützung.

## 1. Februar 2024

### Amazon IVS Broadcast SDK: Android 1.14.1, iOS 1.14.1, Web 1.8.0 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
| --- | --- |
| Web-Broadcast-SDK 1.8.0 | Referenzdokumentation: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference<br>• Die mehrschichtige Kodierung mit Simulcast ist jetzt standardmäßig deaktiviert.<br>• Es wurde ein Problem behoben, bei dem eine Stage-Instance nicht sauber getrennt wurde, wenn eine Stage gelöscht wurde oder wenn ein Teilnehmer vom Server getrennt wurde. Das SDK gibt jetzt ein `STAGE_CONNECTION_STATE_CHANGED` Ereignis mit dem Status `DISCONNECTED` (anstelle von `ERRORED` und dann `CONNECTING` ) aus.<br>• Es wurde ein Problem behoben, bei dem die Veröffentlichung fehlschlägt, wenn die Strategie mit leeren Audio- oder Videospuren aktualisiert wird. |

| Plattform | Downloads und Änderungen |
| --- | --- |
| Android Broadcast SDK 1.14.1 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/android<br>• Die mehrschichtige Kodierung mit Simulcast ist jetzt standardmäßig deaktiviert.<br>• Aktualisiert `libWebRTC` von M108 auf M119.<br>• Es wurden mehrere Abstürze behoben, um die allgemeine Stabilität zu verbessern.<br>• Unterstützung für Stereo-Veröffentlichung hinzugefügt. Dies kann über das `-StageAudioConfiguration` Objekt aktiviert werden.<br>• Es wurde ein Fehler behoben, der zu einem schwarzen Feed von Teilnehmern führte, nachdem sie einer Sitzung beigetreten waren.<br>• Aktualisierte interne `libWebRTC` Referenzen, um Symbolkonflikte zu vermeiden, wenn andere `libWebRTC` Versionen in derselben Hostanwendung enthalten sind. |

| Plattform | Downloads und Änderungen |
| --- | --- |
| iOS-Broadcast-SDK 1.14.1 | Für Echtzeit-Streaming herunterladen: https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip<br><br>Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/ iOS<br><br>• Die mehrschichtige Kodierung mit Simulcast ist jetzt standardmäßig deaktiviert.<br>• Aktualisiert `libWebRTC` von M108 auf M119.<br>• Es wurden mehrere Abstürze behoben, um die allgemeine Stabilität zu verbessern.<br>• Unterstützung für Stereo-Veröffentlichung hinzugefügt. Dies kann über `IVSLocalStageStreamAudioConfiguration` aktiviert werden.<br>• Es wurde ein Absturz beim Aktivieren des reinen Audiomodus für andere Teilnehmer behoben.<br>• Verbesserte TTV und reduzierte Binärgröße. |

## Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64-v8a | 5,223 MB | 13,118 MB |
| armeabi-v7a | 4,524 MB | 9,134 MB |
| x86_64 | 5,418 MB | 13,955 MB |
| 86 x | 5,61 MB | 14,369 MB |

### Broadcast-SDK-Größe: iOS

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64 | 3,350 MB | 7,790 MB |

## 3. Januar 2024

### Amazon IVS Broadcast SDK: Android 1.13.4, iOS 1.13.4, Web 1.7.0 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
| --- | --- |
| Web-Broadcast-SDK 1.7.0 | Referenzdokumentation: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference<br>• Verbesserte time-to-video für Subscriber, die Stufen betreten.<br>• Die `-minAudioBitrateKbps` Eigenschaft wurde entfernt (sie wurde nicht verwendet).<br>• Verbesserte Netzwerkwiederherstellung bei Internetausfällen oder -änderungen. |
| Android Broadcast SDK 1.13.4 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/android<br>• StageAudioConfiguration unterstützt jetzt die Einstellung, ob die Echounterdrückung aktiviert werden soll. |
| iOS-Broadcast-SDK 1.13.4 | Für Echtzeit-Streaming herunterladen: https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip<br>Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/ iOS |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | • Unter iOS haben wir die Audio-Engine sowohl für die Aufnahme als auch für die Wiedergabe verbessert, wobei der Schwerpunkt auf Stabilität und Wiederher stellbarkeit gelegt wurde. Dies verbesser t die Unterstützung für Routenänderungen während der Verwendung, verbessert die Batteriewiederherstellung für Edge-Fälle und reduziert die Menge der Hauptthreadblockie rung.<br>• Es wurde ein Problem behoben, bei dem das Mikrofon möglicherweise aktiv bleibt, auch nachdem es von einer Stufe getrennt wurde, wobei der iOS-Datenschutzindikator aktiviert bleibt. (Das SDK verarbeitete zu diesem Zeitpunkt kein eingehendes Audio.) |

## Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64-v8a | 5,187 MB | 13,025 MB |
| armeabi-v7a | 4,491 MB | 9,056 MB |
| x86_64 | 5,359 MB | 13,829 MB |
| 86 x | 5,553 MB | 14,214 MB |

## Broadcast-SDK-Größe: iOS

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64 | 3,45 MB | 7,84 MB |

# 07. Dezember 2023

## Neue CloudWatch Metriken

Wir haben die Metrik PacketLoss (Phase) in DownloadPacketLoss (Phase) umbenannt. Wir haben auch zusätzliche CloudWatch Metriken für IVS-Echtzeit-Streaming veröffentlicht:

- DownloadPacketLoss (Phase, Teilnehmer)
- DroppedFrames (Phase, Teilnehmer)
- SubscribeBitrate (Stufe, Teilnehmer,MediaType)

Weitere Informationen finden Sie unter Überwachen von IVS-Echtzeit-Streaming.

# 4. Dezember 2023

## Amazon IVS Broadcast SDK: Android 1.13.2 und iOS 1.13.2 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
|---|---|
| Alle Mobilgeräte (Android und iOS) | • Die Konfiguration zur Geräuschunterdrück ung steht Entwicklern zum Aktivieren/Deaktiv ieren für die Veröffentlichung zur Verfügung. |
| Android Broadcast SDK 1.13.2 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/android<br>• Die Ladezeit des Videos (TTV) beim Beitritt zur ersten Stufe einer Sitzung wurde verbessert. |
| iOS-Broadcast-SDK 1.13.2 | Für Echtzeit-Streaming herunterladen: https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip<br>Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/ iOS |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | • Keine Änderungen am Echtzeit-SDK. |

### Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64-v8a | 5,177 MB | 13,01 MB |
| armeabi-v7a | 4,485 MB | 9,045 MB |
| x86_64 | 5,352 MB | 13,808 MB |
| 86 x | 5,547 MB | 14,192 MB |

### Broadcast-SDK-Größe: iOS

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64 | 3,45 MB | 7,82 MB |

## 21. November 2023

### Amazon IVS Broadcast SDK: Android 1.13.1 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
| --- | --- |
| Android-Broadcast-SDK 1.13.1 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.1/android<br>• Es wurde ein Problem behoben, das beim schnellen Verlassen, Loslassen und erneuten Betreten derselben Stufe zu einem Absturz führte. |

### Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
|---|---|---|
| arm64-v8a | 5,177 MB | 13,102 MB |
| armeabi-v7a | 4,485 MB | 9,046 MB |
| x86_64 | 5,353 MB | 13,809 MB |
| 86 x | 5,547 MB | 14,192 MB |

## 17. November 2023

### Amazon IVS Broadcast SDK: Web 1.13.0 und iOS 1.13.0 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
|---|---|
| Alle Mobilgeräte (Android und iOS) | • Aktualisierte Streaming-Optimierungen. Unter anderem erfordert das Feature „Adaptives Streaming: Ebenen-Codierung mit Simulcast “ jetzt eine ausdrückliche Zustimmung und wird nur in neueren Versionen des SDK unterstützt.<br>• Die Stabilität der Stufen wurde verbesser t, indem das Auftreten seltener Abstürze reduziert wurde.<br>• Die Ladezeit des Videos (TTV) beim Beitritt zu einer Phase wurde verbessert.<br>• Die Erfahrung mit Bluetooth-Geräten wurde verbessert.<br>• Die CPU- und Speichernutzung des SDK wurde optimiert und die Bibliotheksgröße reduziert. |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | • Die `StageAudioManager` -Klasse wurde hinzugefügt, mit der sich Audioaufnahme- und Wiedergabeparameter festlegen lassen, einschließlich Voreinstellungen für Sprachkommunikation, Medienwiedergabe und mehr. Einzelheiten finden Sie auf der neuen Seite IVS Broadcast SDK: Mobile Audiomodi.<br>• Es wurde eine neue `requestQualityStats` -Funktion hinzugefügt, um strukturierte Qualitätsereignisse aus WebRTC-Statistiken anzuzeigen.<br>• Es wurde eine neue Funktion zur Aktualisierung der Audiobitrate hinzugefügt. Es wird wie die Videokonfiguration auf `LocalStageStream` -Objekte festgelegt, jedoch über ein neues Audiokonfigurationsobjekt. |

| Plattform | Downloads und Änderungen |
| --- | --- |
| Android Broadcast SDK 1.13.0 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/android<br>• Alle Methoden in der `StageRenderer` -Schnittstelle sind jetzt optional.<br>• Unterstützung für die `Surfaceview` -basierte Vorschau hinzugefügt, um eine bessere Leistung zu erzielen. Die vorhanden en `getPreview` -Methoden in `Session` und `StageStream` geben weiterhin eine Unterklasse von `TextureView` zurück, dies kann sich jedoch in einer zukünftigen SDK-Version ändern.<br>  • Wenn Ihre Anwendung speziell von `TextureView` abhängt, können Sie ohne Änderungen fortfahren. Sie können auch von `getPreview` zu `getPrevie wTextureView` wechseln, um sich auf die eventuelle Änderung der Rückgabew erte des Standardwerts `getPreview` vorzubereiten.<br>  • Wenn Ihre Anwendung `TextureView` nicht speziell erfordert, empfehlen wir für eine geringere CPU- und Speichera uslastung den Wechsel zu `getPrevie wSurfaceView` .<br>• Das SDK implementiert jetzt einen neuen Vorschautyp namens `ImagePrev iewSurfaceTarget` , der mit dem von der Anwendung bereitgestellten Android-S urface-Objekt funktioniert. Es handelt sich nicht um eine Unterklasse von Android View, die eine bessere Flexibilität bietet. |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | • Der Fall, bei dem der `onFrame`-Rückruf für entfernte Teilnehmer zur falschen Zeit mit der falschen Größe aufgerufen wurde, wurde behoben.<br>• `SurfaceSource # getInputSurface` ist jetzt mit `@Nullable` annotiert. Ihr Code sollte es vor der Verwendung überprüfen.<br>• `UserId` und `attributes` wurden zu `ParticipantInfo` hinzugefügt. Die Eigenschaften `UserId` und `attributes` sind in das Token eingebettet und können von Anwendungen über `ParticipantInfo` abgerufen werden, wenn ein Teilnehmer beitritt.<br>• Die Kameraaufnahme und das Rendern der Vorschau sind jetzt standardmäßig auf 720 x 1280 oder die Veröffentlichungsauflösung (je nachdem, welcher Wert höher ist) bei 15 FPS eingestellt. Mit `StageVideoConfiguration # setCameraCaptureQuality` können Sie die Auflösung und/oder die Bildfrequenz anpassen.<br>• `IllegalArgumentException`, das beim Festlegen von Konfigurationseigenschaften ausgelöst wird, enthält nun den bereitgestellten Wert in der Ausnahmemeldung. |

| Plattform | Downloads und Änderungen |
| --- | --- |
| iOS Broadcast SDK 1.13.0 | Für Echtzeit-Streaming herunterladen: https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip<br><br>Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/ iOS<br><br>• Es wurde ein Problem behoben, bei dem das SDK die Videokonfiguration nicht ändert, wenn die Videokonfiguration vor der Veröffentlichung aktualisiert wird.<br>• Die Google-Korrektur für eine LibVPX-Schwachstelle (CVE-2023-5217) wurde integriert. (Beachten Sie, dass das Android-SDK für dieses Problem keine Änderungen erfordert.)<br>• Bei Anwendungen, die andere Bibliotheken verwenden, die `libWebRTC` enthalten, treten keine Konflikte mehr mit dem IVS Broadcast SDK auf.<br>• Alle Methoden im `IVSStageRenderer`-Protokoll sind jetzt als `@optional` markiert.<br>• Von unseren SDKs zurückgegebene Mikrofone und Kameras haben jetzt eine garantierte Sortierreihenfolge, wie in den SDKs selbst dokumentiert.<br>• Mehrere Kameras können jetzt den Wert `true` für ihre `isDefault`-Eigenschaft haben, einen für jede vom Betriebssystem festgelegte Position.<br>• `IVSStageAudioManager` hinzugefügt, das eine präzise Kontrolle über das zugrunde liegende `AVAudioSession` ermöglicht, um |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | eine größere Vielfalt an Anwendungsfällen für die Stufen-Funktionalität zu ermöglichen.<br>• `UserId` wurde `ParticipantInfo` hinzugefügt. |

### Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64-v8a | 5,17 MB | 13,00 MB |
| armeabi-v7a | 4,48 MB | 9,04 MB |
| x86_64 | 5,35 MB | 13,80 MB |
| 86 x | 5,54 MB | 14,18 MB |

### Broadcast-SDK-Größe: iOS

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64 | 3,45 MB | 7,84 MB |

## 16. November 2023

### Zusammengesetzte Aufzeichnung

Dieses neue Feature ermöglicht die Aufzeichnung der zusammengesetzten Ansicht einer IVS-Stufe in einem Amazon-S3-Bucket. Weitere Informationen finden Sie hier:

- Zusammengesetzte Aufnahme – Dies ist eine neue Seite.
- Erste Schritte mit IVS-Echtzeit-Streaming – Es wurden S3-Endpunkte zur Richtlinie in „IAM-Berechtigungen einrichten“ hinzugefügt.

- Service Quotas – Es wurden Aufruf-Kontingente für die neuen Endpunkte hinzugefügt.
- Referenz zur IVS-Echtzeit-Streaming-API – Wir haben 4 StorageConfiguration Endpunkte und 7 Objekte (DestinationDetail, RecordingConfiguration, S3DestinationConfiguration, S3Detail, S3StorageConfiguration, StorageConfiguration,) hinzugefügt StorageConfigurationSummary. Wir haben auch 3 Objekte geändert (Komposition, Ziel, DestinationConfiguration); dies wirkt sich auf die GetComposition Antwort sowie die StartComposition Anfrage und Antwort aus.

# 16. November 2023

## Serverseitige Zusammensetzung

Mit der serverseitigen IVS-Zusammensetzung können Clients die Zusammensetzung und Übertragung einer IVS-Stufe an einen von IVS verwalteten Service verlagern. Die serverseitige Zusammensetzung und RTMP-Übertragung an einen Kanal werden über Endpunkte der IVS-Steuerebene in der Heimatregion der Stufe aufgerufen. Weitere Informationen finden Sie hier:

- Erste Schritte mit IVS-Echtzeit-Streaming – Es wurden SSC-Endpunkte zur Richtlinie im Abschnitt „IAM-Berechtigungen einrichten“ hinzugefügt.
- Verwenden von Amazon EventBridge mit IVS-Echtzeit-Streaming – Wir haben neue Metriken hinzugefügt.
- Serverseitige Zusammensetzung – Dieses neue Dokument enthält eine Übersicht und Anweisungen zur Einrichtung.
- Service Quotas (Echtzeit-Streaming) – Es wurden neue Anrufratenlimits und andere Kontingente hinzugefügt.
- Referenz zur Echtzeit-Streaming-API – Wir haben 8 Zusammensetzungen und EncoderConfiguration Endpunkte und 11 Objekte (ChannelDestinationConfiguration, Zusammensetzung, CompositionSummary, Ziel DestinationConfiguration, DestinationSummary, EncoderConfiguration, EncoderConfigurationSummary, GridConfiguration LayoutConfiguration, und Video) hinzugefügt.

Im IVS-Streaming-Benutzerhandbuch mit niedriger Latenz finden Sie Folgendes:

- Aktivierung mehrerer Hosts in einem IVS-Stream – Es wurde „Broadcasting einer Stufe: Clientseitige im Vergleich zu serverseitige Zusammensetzung“ hinzugefügt und „4. Übertragung der Stufe“ wurde aktualisiert.

# 16. Oktober 2023

## Amazon IVS Broadcast SDK: Web 1.6.0 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
| --- | --- |
| Web-Broadcast-SDK 1.6.0 | Referenzdokumentation: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference<br>• Verbesserte Time-To-Video (TTV).<br>• `maxAudioBitrate` -Konfiguration hinzugefügt, die bis zu 128 kbit/s an Mono- oder Stereo-Audiokanälen unterstützt. |

# 12. Oktober 2023

## Neue CloudWatch Metriken und Teilnehmerdaten

Wir haben CloudWatch Metriken für IVS-Echtzeit-Streaming veröffentlicht. Weitere Informationen finden Sie unter Überwachen von IVS-Echtzeit-Streaming.

Dem Teilnehmer-API-Objekt wurden auch sechs Felder hinzugefügt: `browserName`, `browserVersion`, `ispName`, `osName`, `osVersion` und `sdkVersion`. Dies wirkt sich auf die GetParticipant Antwort aus. Siehe die API-Referenz zu IVS-Echtzeit-Streaming.

# 12. Oktober 2023

## Amazon IVS Broadcast SDK: Android 1.12.1 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
| --- | --- |
| Android-Broadcast-SDK 1.12.1 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.12.1/android |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | • Es wurde ein Fehler behoben, bei dem das Aufrufen von `BroadcastSession.setListener` zu einem Fehler führte. |

### Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64-v8a | 5,853 MB | 16,375 MB |
| armeabi-v7a | 4,895 MB | 10,803 MB |
| x86_64 | 6,149 MB | 17,318 MB |
| 86 x | 6,328 MB | 17,186 MB |

## 14. September 2023

### Amazon IVS-Broadcast-SDK: Web 1.5.2 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
| --- | --- |
| Web-Broadcast-SDK 1.5.2 | Referenzdokumentation: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference<br>• Es wurde ein Fehler behoben, der das erneute Veröffentlichen mit `refreshStrategy` verhinderte, wenn der Status „Veröffentlicht“ in einen `ERRORED`-Status übergeht. |

# 23. August 2023

## Amazon IVS Broadcast SDK: Web 1.5.1, Android 1.12.0 und iOS 1.12.0 (Echtzeit-Streaming)

| Plattform | Downloads und Änderungen |
| --- | --- |
| Web-Broadcast-SDK 1.5.1 | Referenzdokumentation: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference<br>• Es wurde ein Fehler mit internen Maybe-Typen auf TypeScript 5 behoben.<br>• Bessere Erkennung für Simulcast-Unterstützung hinzugefügt.<br>• Zwei Race-Bedingungen mit `refreshStrategy` beim Versuch zu veröffentlichen wurden behoben.<br>• Eine Race-Bedingung mit `refreshStrategy` beim Versuch, Teilnehmer zu aktualisieren, um sie zu abonnieren, wurde behoben. |
| Alle Mobilgeräte (Android und iOS) | • Es wurde ein seltenes Problem behoben, bei dem die Veröffentlichungsaktion nie abgeschlossen wurde.<br>• Die Stabilität der Stufen wurde verbessert, indem das Auftreten seltener Abstürze reduziert wurde.<br>• Die Stabilität der Stufen wurde verbessert, indem Probleme mit Race-Bedingungen behoben wurden, die aufgrund des schnellen Beitritts oder Austritts entstanden sind.<br>• Eine neue `setOnFrameCallback`-Methode wurde für `ImageDevice` hinzugefügt. Auf diese Weise kann |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | beobachtet werden, wie die Frames das Gerät selbst durchlaufen, was einen Einblick in das Seitenverhältnis der aktuellen Bilder ermöglicht. Diese Methode kann auch verwendet werden, um zu erkennen, wann der erste Frame für einen Remote-Te ilnehmer in einer Stufe gerendert wird. |
| Android-Broadcast-SDK 1.12.0 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/android<br>• Android 9 wird jetzt unterstützt.<br>• Verbesserte CPU-Auslastung und -Leistung. |
| iOS-Broadcast-SDK 1.12.0 | Für Echtzeit-Streaming herunterladen: https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip<br>Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/ iOS<br>• Die Signatur von `IVSDeviceDiscovery.createAudioSourceWithName` wurde korrigiert und gibt jetzt `IVSCustomAudioSource` statt `IVSCustomImageSource` zurück. |

## Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64-v8a | 5,853 MB | 16,375 MB |
| armeabi-v7a | 4,895 MB | 10,803 MB |
| x86_64 | 6,149 MB | 17,318 MB |

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
|---|---|---|
| 86 x | 6,328 MB | 17,186 MB |

### Broadcast-SDK-Größe: iOS

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
|---|---|---|
| arm64 | 5,06 MB | 10,92 MB |

## 7. August 2023

### Amazon IVS Broadcast SDK: Web 1.5.0, Android 1.11.0, und iOS 1.11.0

| Plattform | Downloads und Änderungen |
|---|---|
| Web-Broadcast-SDK 1.5.0 | Referenzdokumentation: https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference<br>• Simulcast hinzugefügt – Wenn aktiviert, ermöglicht dieses Feature dem Publisher, Videoebenen mit hoher und niedriger Qualität zu senden. Subscriber wählen automatisch ihre optimale Qualität auf der Grundlage ihrer Netzwerkbedingungen aus. Siehe Optimieren von Medien. |
| Alle Mobilgeräte (Android und iOS) | Simulcast hinzugefügt – Wenn aktiviert, ermöglicht dieses Feature dem Publisher, Videoebenen mit hoher und niedriger Qualität zu senden. Subscriber wählen automatisch ihre optimale Qualität auf der Grundlage ihrer Netzwerkbedingungen aus. Siehe „Layered Encoding mit Simulcast aktivieren/deaktivieren“ |

| Plattform | Downloads und Änderungen |
| --- | --- |
| | in den Anleitungen zum Android- und iOS-Broadcast-SDK. |
| Android-Broadcast-SDK 1.11.0 | Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/android<br>• Es wurde ein Problem behoben, bei dem das Erstellen vieler Stufen letztendlich zu einem Absturz führte. (Die genaue Anzahl der Stufen hängt vom Gerät ab.) |
| iOS-Broadcast-SDK 1.11.0 | Für Echtzeit-Streaming herunterladen: https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip<br>Referenzdokumentation: https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/ iOS<br>• Korrigierte die Signatur von `IVSDeviceDiscovery.createAudioSourceWithName` zur Rückgabe von `IVSCustomAudioSource` statt `IVSCustomImageSource`. |

## Broadcast-SDK-Größe: Android

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
| --- | --- | --- |
| arm64-v8a | 5,811 MB | 16,186 MB |
| armeabi-v7a | 4,857 MB | 10,646 MB |
| x86_64 | 6,108 MB | 17,122 MB |
| 86 x | 6,289 MB | 16,994 MB |

### Broadcast-SDK-Größe: iOS

| Architektur | Komprimierte Größe | Unkomprimierte Größe |
|---|---|---|
| arm64 | 5,030 MB | 10,810 MB |

# 7. August 2023

## Streaming in Echtzeit

Mit Amazon Interactive Video Service (IVS)-Echtzeit-Streaming können Sie Live-Streams mit einer Latenz bereitstellen, bei denen, vom Host bis zum Zuschauer, die Latenz unter 300 Millisekunden liegen kann.

Diese Version enthält wichtige Änderungen an der Dokumentation. Die Landingpage zur IVS-Dokumentation hat jetzt separate Abschnitte für Echtzeit-Streaming und Streaming mit niedriger Latenz. Jeder Abschnitt hat sein eigenes Benutzerhandbuch und eine eigene API-Referenz. Einzelheiten zur Dokumentation finden Sie in der Dokumentenhistorie (für Änderungen an der Dokumentation sowohl für Echtzeit und niedriger Latenz). Für Echtzeit-Streaming beginnen Sie mit dem Benutzerhandbuch für IVS-Echtzeit-Streaming und der Referenz zur IVS-Echtzeit-Streaming-API.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.